TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TOMMI TUOMINEN

HTML-SIVUJEN SISÄLLÖNTUOTTAMISEN OPTIMOINTI

Diplomityö

Tarkastaja: professori Pekka Loula
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 6. touko-kuuta 2015

TIIVISTELMÄ

TOMMI TUOMINEN: HTML-sivujen sisällöntuottamisen optimointi
Tampereen teknillinen yliopisto
Diplomityö, 60 sivua
Joulukuu 2015
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Tietoliikennetekniikka
Tarkastaja: professori Pekka Loula

Avainsanat: foreammatti, html, css, optimointi, minimointi, sisällöntuotanto

ForeAmmatti-tietojärjestelmän datasisältö oli kasvanut vuosien mittaan merkittävästi eikä tehokkuuden parantamiseen ollut käytetty aiemmin riittävästi resursseja. HTML-sivujen dynaamisen sisällön tuottaminen oli aina tehty nopeimmalla ja helpoimmalla tavalla, koska ForeAmmatin kehitystyö on ollut erittäin nopeaa. Uusia ominaisuuksia ja sisältöä on lisätty vauhdilla, eikä ratkaisujen tehokkuutta ole aina ehditty pohtia. Tämän johdosta loppukäyttäjälle siirrettävät HTML-sivut sekä niiden resurssit olivat usein sekä tiedostokooltaan että merkkimäärältään valtavan suuria.

Tämän diplomityön tärkeimmäksi muutostarpeeksi osoittautui nimenomaan ammatti- ja aluevalitsimien suunnitteleminen uudelleen. Näiden tekstisisältö on oikeastaan staattista, mutta vaihtoehtojen määrä on käyttäjäkohtainen sekä järjestykseltään käyttäjän muokattavissa. Nykyinen tapa ei ollut tarpeeksi tehokas. Valitsimien toteuttaminen uudella tavalla pienensi HTML-sivujen merkkimääriä huomattavasti, lähes 90 prosenttia. HTML-sivujen lisäksi optimoitiin myös muut sivuston tarvitsemat resurssit ensin sisällön osalta ja sen jälkeen parannettiin vielä niiden jakelua. Resurssit minimoitiin ja jaettiin pakattuina mahdollisimman pienessä koossa.

Lisäksi tehtiin myös paljon muita pienempiä optimointeja. Tärkeimpinä istunnon koon pienentäminen sen luontiajan nopeuttamiseksi sekä yhä lisääntyvien robottikäyttäjien myötä estettiin niitä lataamasta tarpeettomia resursseja palvelinkuormituksen vähentämiseksi. Saavutetut tulokset olivatkin erittäin merkittäviä. ForeAmmatin etusivun latauskoko pienentyi yhteensä noin 80 prosenttia ja latausaika noin 75 prosenttia, kun nettiyhteyden nopeus ei ollut rajoittavana tekijänä. Hitailla yhteyksillä hyödyt ovat vielä suurempia. Istunnon luontiaika nopeutui yli 90 prosenttia ja palvelin pystyy käsittelemään nyt yli kolminkertaisen määrän samanaikaisia käyttäjiä entiseen verrattuna.

ABSTRACT

TOMMI TUOMINEN: Optimizing content production of the HTML pages
Tampere University of Technology
Master of Science Thesis, 60 pages
December 2015
Master's Degree Programme in Information Technology
Major: Telecommunications Engineering
Examiner: Professor Pekka Loula

Keywords: foreammatti, html, css, optimizing, minimization, content production

Data content of the ForeAmmatti information system had been significantly increasing in the course of the last few years and improving its performance has not been given enough resources. The dynamic production of the content in the HTML pages had always been done with the fastest and easiest way, because the development work of ForeAmmatti has been very fast. New features and content have been added with speed and the performance of the solutions have not received major consideration. Due to that, HTML pages and their resources, which are transferred to the end user, were often enormously big considering both the file size and the amount of characters.

The most important modification need this thesis specifically revealed was redesigning the occupation and region selectors. The text content of those is actually static, but the amount of choices is user dependent and the order is editable by user. The current method is not efficient enough. Coding of the selectors with the new way reduced character amount of the HTML pages significantly, almost 90 percent. Along with the HTML pages also the other resources of the web site were optimized. First by the content and then its delivering. Resources were minified and delivered compressed to as small a size as possible.

Additionally, also many other smaller optimizations were done. The most important things were the size reduction of the session to speed up its creation time and the reduction of impact for the server load from the ever increasing robot user hits by preventing them of downloading unnecessary resources. The results obtained were very significant. The download size of the front page of ForeAmmatti was reduced by a total of about 80 percent and the downloading time about 75 percent, while internet connection speed was not a limiting factor. On slower connections, the benefits are to be even bigger. Session creation time speeded up over 90 percent and the server can now handle more than three times as much concurrent users as earlier.

ALKUSANAT

Tämä diplomityö on laadittu tiiviissä yhteistyössä Foredata Oy:n kaikkien työntekijöiden kanssa. Haluankin kiittää erityisesti toimitusjohtaja Jari Järvistä, jonka ansiosta koko tämän työn tekeminen oli ylipäätään mahdollista. Hänen kanssaan yhdessä loimme tälle diplomityölle otolliset olosuhteet, jotta tutkittavasta asiasta saadaan erittäin paljon hyötyä sekä yrityksemme omaan että asiakkaidemme käyttöön.

Lisäksi haluan kiittää arvokkaasta testausavusta ja ongelmien ratkaisemisesta myös Foredata Oy:n muita työntekijöitä, kehitysjohtaja Mikael Andolinia sekä ammattikorkeakoulututkintoni aikaista ystävääni Database Administrator Ilkka Vatajaa. Suuret kiitokset kaikesta saamastani henkisestä tuesta kuuluvat myös avovaimolleni sekä vanhemmilleni.

Harjavallassa, 22.11.2015

Tommi Tuominen

Tommi Tuominen

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

LYHENTEET JA MERKINNÄT

AJAX	Asynchronous JavaScript and XML
CDN	Content Delivery Network
CSS	Cascading Style Sheets
DOM	Document Object Model, dokumenttioliomalli
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure, joskus myös HTTP over TLS tai HTTP over SSL
JSF	JavaServer Faces
JVM	Java Virtual Machine
URL	Uniform Resource Locator, verkkosivun osoite
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language

1. JOHDANTO

Tässä diplomityössä on tarkoitus parantaa loppukäyttäjän kokemaa ForeAmmatti-tietojärjestelmän käyttökokemusta. Foredata Oy:n kehittämää ForeAmmatti-tietojärjestelmää alettiin kehittää syksyllä 2010 yhdessä minun, Ilkka Vatajan sekä Jari Järvisen voimin. Kehitystyön oli suunniteltu olevan minun [41] ja Ilkan [42] opinnäytetyö Satakunnan ammattikorkeakouluun. Silloin insinöörityötkinnon suuntautumisvaihtoehtoni oli ohjelmistotekniikan alalle, joten työ keskittyi pääasiassa sivuston ulkoasun ja toimintojen luomiseen sekä bisneslogiikan koodaamiseen. Diplomi-insinöörityötkintoni pääaine on nyt tietoliikennetekniikka, joten myös tämä diplomityö keskittyy nyt sille puolelle. Toinen merkittävä syy on tietysti käyttäjämäärien suuri lisääntyminen viime aikoina. ForeAmmatin ulkoasu on näiden muutaman vuoden aikana uudistunut jo useammankin kerran ja samoin palvelinta on vaihdettu jo kahteen otteeseen. Nyt onkin siis paikallaan tehdä kaikesta mahdollisimman kevyttä tiedonsiirron ja palvelinkuorman näkökulmasta, jotta ForeAmmatti pystyy jatkossakin palvelemaan entistä suurempia kävijämääriä vaivattomasti.

ForeAmmatin kaikille avoimen osuuden yhtenä päätavoitteena on lyhentää työnhakuun käytettyä aikaa. Se palvelee myös niitä lisätietoa etsiviä, jotka suunnittelevat alanvaihtoa tai pohtivat oman osaamisensa ja koulutustasonsa riittävyyttä. Palvelusta löydät vastauksia muun muassa seuraaviin kysymyksiin: [18]

- Kuinka paljon eri ammateissa on avoimia työpaikkoja? Työpaikkailmoituksia jättävien tulee ilmoittaa joko yleisesti tunnettu ammattinimike tai valita suoraan virallisen ammattiluokituksen [2] ammatti.
- Millä alueella työpaikat ovat? ForeAmmatti louhii työpaikkailmoituksista esiin myös tiedon työpaikan sijaintikunnasta, ja nämä tiedot voidaan esittää käyttäjälle sekä listamuodossa että havainnollistavana karttaesityksenä.
- Mitä osaamista työnantajat työnhakijoilta haluavat? Voimme analysoida yli 1,38 miljoonan tietokannastamme löytyvän työpaikkailmoituksen tekstit ja löytää niistä ne osaamiset, jotka toistuvat monissa ilmoituksissa.
- Kuinka kovaa kilpailu työpaikoista on nyt ja lähitulevaisuudessa? Edellisen kuukauden päätyttyä tilanne päivittyy yleensä noin kolmen viikon jälkeen Toimiala Online -palveluun [40], josta löytyy myös työ- ja elinkeinoministeriön tuottama työnvälitystilasto. Lähitulevaisuuden ennusteen (noin kolmen vuoden päähän) laativat Foredata Oy:n asiantuntijat käyttäen pääasiassa melko luotettavasti ennustettavia parametreja, kuten eläkepoistumaa ja oppilaitosten tutkintotuotosta.

ForeAmmatin päätarkoitus on toki edelleen sama kuin edellisen opinnäytteen valmistumisenkin aikaan, neljä ja puoli vuotta sitten, mutta sisältö on toki aivan erinäköistä kuin

silloin. Informatiivisen datan määrä on kasvanut huomattavasti ja kasvaa koko ajan. Työpaikkailmoituksia jätetään päivittäin ja tilastotietoja julkaistaan joka kuukausi. Viimeinen suuri ulkoasun muutos tapahtui tammikuussa 2015, kun ForeAmmatin kaikille avoin maksuton puoli saatiin tehtyä mobiiliystävälliseksi ja erillisen mobiilisovelluksen käytöstä päätettiin luopua kokonaan. Näin ollen tämän jatkuvasti iteroituvan ympäristön seuraava kehitysaskel on tietysti suorituskyvyn parantaminen

Tähän asti jatkuneen ja edelleen koko ajan jatkuvan kehitystyön rinnalla on keskitytty lähinnä ohjelmakoodin helppouteen ja tietokantojen toimivuuteen. Viimeisen vuoden aikana painotusta on jatkuvasti siirretty yhä enemmän ja enemmän tehokkuuden parantamiseen jatkuvasti lisääntyneen käytön ja kasvaneen käyttäjämäärän johdosta. Ohjelmakoodia on nopeutettu ja tietokantoja optimoitu nyt siinä määrin, että niistä prosentuaalisesti saatava hyöty on ollut suurinta. Jatkossa saatava hyöty suhteessa työtuntien määrään putoaa, mikä onkin syynä siihen, että kehitystyön pääpaino siirtyy nyt uusille urille.

Seuraavaksi on tarkoitus tutustua itse XHTML (eXtensible Hypertext Markup Language) -sivujen ja näihin liittyvien ulkoisten resurssien sisällön optimointiin sekä tietoliikenteen kehittämiseen. Näiden asioiden kehitystyön etenemistä seurataan tässä diplomityössä. Työn pääasiallinen tarkoitus on tutkia erilaisia vaihtoehtoja ja menetelmiä, miten loppukäyttäjälle siirrettävän datan määrää voidaan laskea lopulliseen näkyvään sisältöön vaikuttamatta. Samalla tutkitaan, onko liikenne mahdollisimman hyvin optimoitua ja loppukäyttäjän kokemat viiveet palvelun käytössä riittävän pieniä.

2. TUTKIMUSYMPÄRISTÖ JA ONGELMAKOHTIEN HAVAITSEMINEN

Tämän työn aikana palvelimena toimii samanlainen virtuaalipalvelin, jossa ForeAmmattikin pyörii. Palvelin on Jelastic-pilvipalveluun perustuva ja vuokrattu Jelasticin suomalaiselta palveluntarjoajalta Planeetta Internet Oy:ltä. Palvelimessa toimii 64-bittinen (x86-64) CentOS 6.6 -käyttöjärjestelmä, joka käyttää Linuxin kernelin versiota 2.6.32-042stab105.14. Palvelimessa on kaksi fyysistä monisäikeistystä tukevaa kuusiytimistä Intel Xeon E5645 -suoritinta, joten yhteensä käytettävissä on 24 suoritinydintä. Palvelimen käytettävissä on 32 gigatavua keskusmuistia. Javan virtuaalikoneelle (JVM, Java Virtual Machine) on käynnistyksen yhteydessä varattu kaksi gigatavua ja tietokantapalvelimelle 512 megatavua keskusmuistia. Näillä asetuksilla prosessien kokonaismuistinkäyttö on liki main neljä gigatavua. Asetettuihin raja-arvoihin lisätään muun muassa sovelluksien itsensä tarvitsema muistimäärä sekä monia erilaisia pienempiä osakokonaisuuksia, kuten välimuisteja ja puskureita pääasiassa levyoperaatioita varten. Verkkoyhteytenä toimii tavallinen 100/100 megabitin kiinteä yhteys. Laitteisto, sovellukset ja asetukset ovat täysin identtisiä ForeAmmatin tuotantopalvelimen kanssa.

Asiakkaina työssä toimivat sekä oma pöytäkoneeni yksittäisiin testeihin että kehityskäytössämme oleva palvelinlaitteisto. Pöytäkoneessani on 64-bittinen Windows 7 -käyttöjärjestelmä ja palvelimessa on 64-bittinen (x86-64) Ubuntu 14.04.3 LTS -käyttöjärjestelmä, jonka kernelin versio on 3.13.0-53-generic. Kummassakin laitteessa on kahdeksan gigatavua keskusmuistia sekä neljännen sukupolven Intel Core -prosessorit. Pöytäkoneessa on monisäikeistystä tukeva neliytiminen i7-4770K-prosessori ja palvelimessa ilman monisäikeistystä toimiva neliytiminen i5-4278U-prosessori. Molemmat laitteet sijaitsevat samassa lähiverkossa, joka on kytketty Internetiin käytännössä 220/13 megabitin kaapelilaajakaistalla. Kaikki verkkoyhteyttä vaativat toimenpiteet tapahtuvat siis julkisen Internetin yli, yleisesti arkipäivisin virka-aikana.

2.1 Mittaus- ja testaustyökalut

Seuraavaksi käydään hieman läpi työkaluja, joita tarvitaan mittausten suorittamiseen. Kaikissa yleisimmissä selaimissa on nykyään sisäänrakennetut kehittäjätyökalut, joilla pääsee jo melko pitkälle. Eri selaimien välillä on tietysti melko suuriakin eroja, miten kevyesti nämä työkalut toimivat, ja miten informatiivisesti ne asiat esittävät. Ehdottomasti eniten itse olen käyttänyt Chromen kehittäjätyökalua, jossa mielestäni elementtien, verkkoliikenteen ja virhekonsolin käyttö ja ulkoasu on selkeintä. Chromeen saa nykyään jo Googlen PageSpeed Insights -liitännäisen [32], joka analysoi sivun ja kertoo sen jälkeen parannusehdotuksia liittyen HTML (Hypertext Markup Language) -sivuun, Ja-

vaScript-koodiin, kuvien kokoihin ja mittoihin sekä tiedostojen otsikkotietojen sekä selaimen välimuistin toimintaan. Chromen verkkoliikenteestä kertova välilehti esittää myös selkeimmän aikajanan, mikä resurssi joutuu odottamaan muita, kauanko sen lataaminen kestää ja missä vaiheessa dokumentin lataus on. PageSpeed Insights -sivustoa on aiemminkin käytetty vastaavanlaisien optimointien apuvälineenä, kuten ovat tehneet muun muassa Tero Elonen [13] sekä Heidi Virta [43] insinööritöissään.

Firefoxin kehittäjätyökalun verkkoliikenteestä kertova välilehti esittää mielestäni selkeämmin käytettiinkö resurssin lataamiseen salattua vai salaamatonta yhteyttä selkeällä lukkokuvalla, koska pitkässä resurssien listassa http:n (Hypertext Transfer Protocol) ja https:n (HTTP Secure) erottaminen toisistaan on hieman hankalaa. Molemmissa selaimissa pystyy myös välimuistin käytön estämään kehittäjätyökalun ollessa auki, mutta Firefoxissa se pitää hakea aina eri välilehdeltä. Myöskään dokumentin tapahtumista selain ei kerro mitään, eikä näytä ollenkaan niitä resursseja, jotka ladataan välimuistista. Firefoxiin saa kuitenkin Firebug-liitännäisen [17], johon puolestaan saa CSS Usage -lisäosan [9], jota käytän myöhemmin ylimääräisten tyylimääritysten löytämiseen. Firebug näyttää muun muassa resurssien tarkan koon tavuina, sekä vastaanotettuna määränä että vastauksen sisältämän hyötykuorman kokona. Näiden välinen erotus on siis tiedonsiirtopakettien otsikkotietoja. Firebug sisältää myös erinomaisen DOM-rakenteen (Document Object Model, dokumenttioliomalli) tutkijan, josta nähdään esimerkiksi JavaScript-muuttujat ja niiden arvot helposti yhdeltä ruudulta. DOM on kuvaus siitä, miten dokumentti ymmärretään rakenteiseksi olioksi, jolla on ominaisuuksia [30]. Käytännössä siis DOM voidaan ajatella puurakenteena, jossa sisaruselementit ovat vierekkäin isäntäelementtinsä alla. Jokaisen elementin attribuutit ja niiden arvot ovat taas elementin alla ja niin edelleen.

Yksittäisten tietoliikennepakettien sisältöön, otsikkotietoihin ja pakettien määriin päästään käsiksi esimerkiksi Wireshark-ohjelman avulla. Palvelinta voidaan ruuhkauttaa jatkuvilla yhtäaikaisilla sivupyynnöillä esimerkiksi Siege-työkalun avulla, kuten Elonen [13] ja Virta [43] ovat tehneet. Minulla oli jo asennettuna ApacheBench-työkalu, joka toimii käytännössä samalla tavalla kuin Siege. Palvelimen kuormitusta mitataan sysstat-pakettiin kuuluvan sar-työkalun avulla Linux-käyttöjärjestelmässä, samaan aikaan kun ApacheBench-testausohjelma lähettää jatkuvasti uusia sivupyyntöjä palvelimelle. Lisäksi käytetään Load Impact -kuormitustesteriä [31], jonka ilmaisversio simuloi enintään sataa yhtäaikaista käyttäjää. Käyttäjien määrä kasvaa vähitellen sataan viiden minuutin aikana. Elonen oli työssään havainnut samaisen testin hyväksi [13], mutta Virran työssä kohdattiin ylitsepääsemättömiä palomuuriongelmia [43].

2.2 Etusivun resurssit

Tutkitaan ensin alkutilannetta selaimien kehittäjätyökaluilla ja niiden lisäosilla, joilla nähdään tiedostojen lataamiseen kuluvat ajat sekä niiden tarkat koot. Lähes kaikki tiedostot, joita koko sivustolla tarvitaan, ladataan alkutilanteen hetkellä jokaisella XHTML-sivulla. Tästä syystä mittauksissa keskitytään vain pelkkään etusivuun ja sen sisältämään

dataan. Latausaikojen kestot mitataan tulosten luotettavuuden varmistamiseksi selaimen välimuisti pois käytöstä kymmenen kerran keskiarvona.

Taulukko 1. *Etusivun resurssien koot ja ajat yhteensä alkutilanteessa.*

Sisällön tyyppi	**Koko yhteensä**	**Aika yhteensä**
XHTML-sivu (1 kpl)	420 998 tavua	431,7 ms
Pyydetty CSS (2 kpl)	30 162 tavua	75,3 ms
Fontit edellisiin (2 kpl)	43 400 tavua	80,3 ms
Pyydetty JS (11 kpl)	332 210 tavua	762,8 ms
Sisältö edellisiin (7 kpl)	112 006 tavua	207,0 ms
Kuvat (6 kpl)	9 129 tavua	126,0 ms
Yhteensä (29 kpl)	**947 905 tavua**	**1 683,1 ms**

Olen jakanut sivun sisällön kuuteen eri ryhmään taulukon 1 mukaisesti. Ensimmäisenä on itse XHTML-sivu, jonka lataaminen on aina se, mistä koko yksittäisen sivun latausprosessi alkaa. Vasta kun XHTML-sivua aletaan jäsentämään, sen sisällöstä löydetään viittaukset muihin resursseihin, jotka se määrää lataamaan. Muita resursseja voidaan ladata useammalla yhteydellä samanaikaisesti, mutta niiden jäsentäminen tapahtuu yksitellen. Pyydetyt tyylitiedostot (CSS, Cascading Style Sheets) tarkoittavat ForeAmmatin omaa tyylitiedostoa sekä Roboto Slab -fonttia, jonka pyyntö ohjataan Googlen palvelimelle. Fontit ovat tietysti edellä mainitut fontit kahdella eri paksuudella.

Pyydetyt JavaScript-komentosarjat (JS) ovat muun muassa itse tehtyä JavaScript-koodia, jQuery-kirjasto, Highcharts-grafiikkakirjaston kolme tiedostoa, Google Maps -liitännäisen latauskoodi, Google Analytics -lisäosa, kaksi Facebook-lisäosaa sekä Inspectlet-lisäosa ja yksi sivuston toiminnan ja responsiivisuuden takaava apukirjasto. Seuraava ryhmä on JavaScript-koodien lataama uusi sisältö, jolla tarkoitetaan muun muassa Google Maps -lisäosan JavaScript-tiedostoja sekä sen ja muiden lisäosien autentikointeja. Kuvat sisältävät luonnollisesti sivulla näkyviä kuvatiedostoja.

Taulukko 2. *Tapahtumat ja niiden hetket sivun latauksen aloittamisesta.*

Tapahtuman tyyppi	**Keskiarvo**	**Keskihajonta**
DOMContentLoaded-tapahtuma	745,6 ms	75,4 ms
Load-tapahtuma	806,4 ms	109,6 ms
Lataaminen päättyy	837,6 ms	110,6 ms

Resurssien samanaikaisesta lataamisesta johtuen itse sivun lataaminen ei silti kestä niin kauaa kuin taulukon 1 aika yhteensä -summasarakkeesta voisi kuvitella, vaan todellisuudessa sivun lataamiseen kuluva aika selviää taulukosta 2. DOMContentLoaded-tapahtuma laukeaa siis keskimäärin 745,6 millisekunnin ja Load-tapahtuma 806,4 millisekunnin kuluttua sivun latauksen aloittamisesta. DOMContentLoaded-tapahtuma laukeaa sil-

loin, kun dokumentti itse on kokonaisuudessaan ladattu ja jäsennetty selaimen toimesta [11]. Tätä tapahtumaa varten ei tarvitse odottaa tyylitiedostojen, kuvien ja muiden resurssien latautumista, paitsi jos dokumentissa on synkronisesti ladattavia JavaScript-komentosarjoja tyylitiedoston lataamisen jälkeen, kuten tässä tapauksessa on. Tässä tilanteessa myös CSS-tyylitiedoston pitää latautua synkronisten JavaScript-komentosarjojen ohella, ennen kuin DOM-puu voidaan rakentaa valmiiksi. Load-tapahtuma laukeaa, kun kaikki dokumentin vaatimat synkronisesti ladattavat resurssit on ladattu kokonaisuudessaan ja dokumentin jäsentämisen pysäyttävät resurssit on jäsennetty. Resurssien lataaminen voi myös alkaa viivästetysti, tai tapahtua asynkronisesti, jolloin niiden lataaminen voi päättyä vielä Load-tapahtuman jälkeen. Tällaisia ovat muun muassa Inspectlet-lisäosa sekä Google Maps -lisäosan JavaScript-tiedostot. Siksi lataaminen päättyy keskimäärin 837,6 millisekunnin kuluttua sivun latauksen aloittamisesta.

Koska itse XHTML-dokumentin lataamiseen kuluva aika on kokonaisuudesta suhteellisen suuri osa (431,7 ms / 837,6 ms ≈ 51,5 %) kuten myös sen koko (420 998 tavua / 947 905 tavua ≈ 44,4 %), niin voidaan olettaa, että myös sen jäsentämiseen kuluu suhteellisen paljon aikaa. Koska dokumentin jäsentämisen taustalla resursseja ladataan useita samaan aikaan, tässä tapauksessa nopealla nettiyhteydellä, niin DOMContentLoaded-tapahtuman ja Load-tapahtuman ajat ovat suhteellisen lähellä toisiaan. Tavoitteena on siis nopeuttaa sekä dokumentin lataamista että jäsentämistä, jotta DOMContentLoaded-tapahtuma laukeaisi aiemmin ja sivu näyttäisi ulkoasun ja tekstin osalta valmiilta. Ensimmäiset dokumenttiin liittyvät alustustoimenpiteet voidaan suorittaa esimerkiksi JavaScriptin avulla DOMContentLoaded-tapahtuman hetkellä.

2.3 Yhteyden muodostamis- ja tiedonsiirtoajat

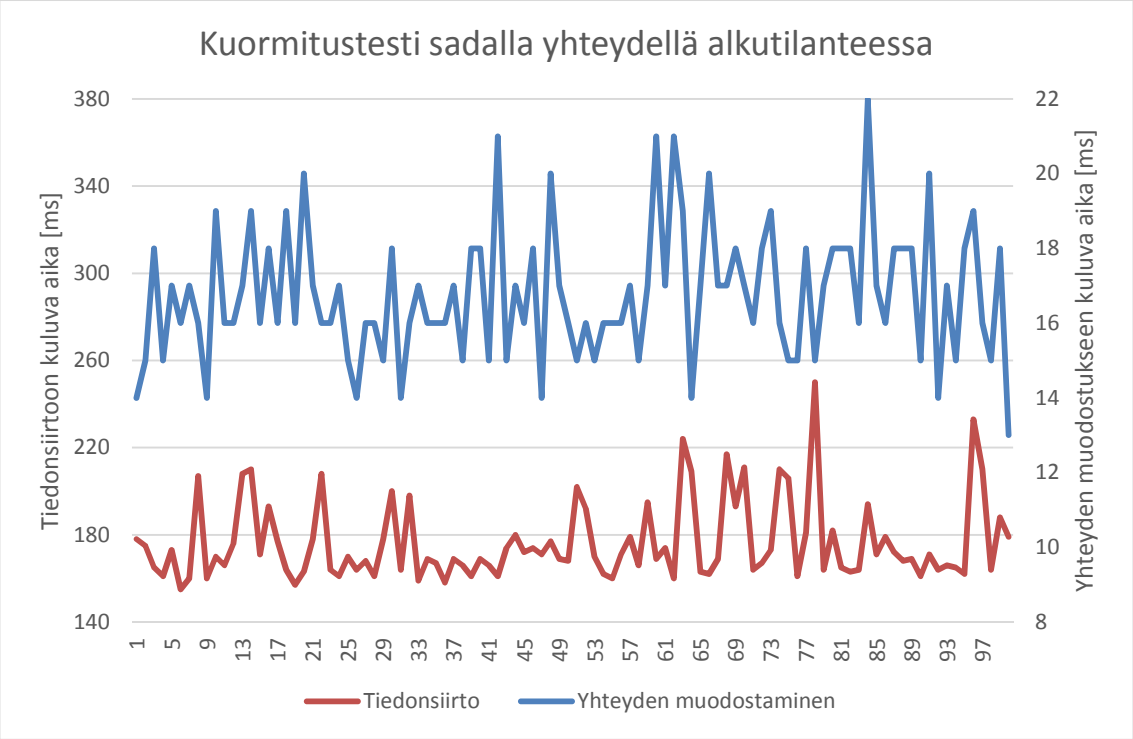
Tässä mittauksessa tutkitaan koneellisesti, kauanko yhdeltä käyttäjältä kuluu aikaa yhteyden muodostamiseen palvelimelle ja miten pitkään dokumentin siirtäminen kestää. Asetetaan ApacheBench-työkalu lähettämään jatkuvasti uusia sivupyyntöjä palvelimelle. Tässä mittauksessa simuloidaan pelkästään yhtä samanaikaista käyttäjää ja mittauksia tehdään sata kappaletta. Suurempi samanaikaisten käyttäjien määrä vääristäisi tuloksia, koska nyt halutaan selvittää vaikutus nimenomaan yhteen käyttäjään. ApacheBench lataa vain itse HTML-dokumentin, ei siihen liittyviä muita tiedostoja.

Taulukko 3. *Yhteyden muodostamisen ja tiedonsiirron kesto alkutilanteessa.*

	Minimi	Mediaani	Keskiarvo	Maksimi	Keskihajonta
Yhteyden muodostaminen	13,00 ms	16,50 ms	16,75 ms	22,00 ms	1,77 ms
Tiedonsiirto	155,0 ms	169,5 ms	176,5 ms	250,0 ms	18,5 ms

Tarkastellaan seuraavaksi ApacheBenchin antamaa kuormitustestin tulosta, jonka tulokset sadan sivupyynnön jälkeen ovat taulukon 3 mukaiset. Todetaan, että yhteyden muodostamiseen kuluva aika vaikuttaa melko pieneltä. Mediaani on 16,50 ms ja keskiarvo

puolestaan 16,75 ms sekä keskihajonta 1,77 ms. Vaihteluväli näyttää olevan 13 - 22 ms. Tiedonsiirtoon kuluvan ajan mediaani on 169,5 ms ja keskiarvo puolestaan 176,5 ms sekä keskihajonta 18,5 ms. Huomion arvoista on se, että tämä testi ainoastaan lataa dokumentin jäsentämättä sitä. Voidaan siis olettaa, että itse XHTML-sivun tuleva pienentäminen näkyisi hyvin tässä testissä.



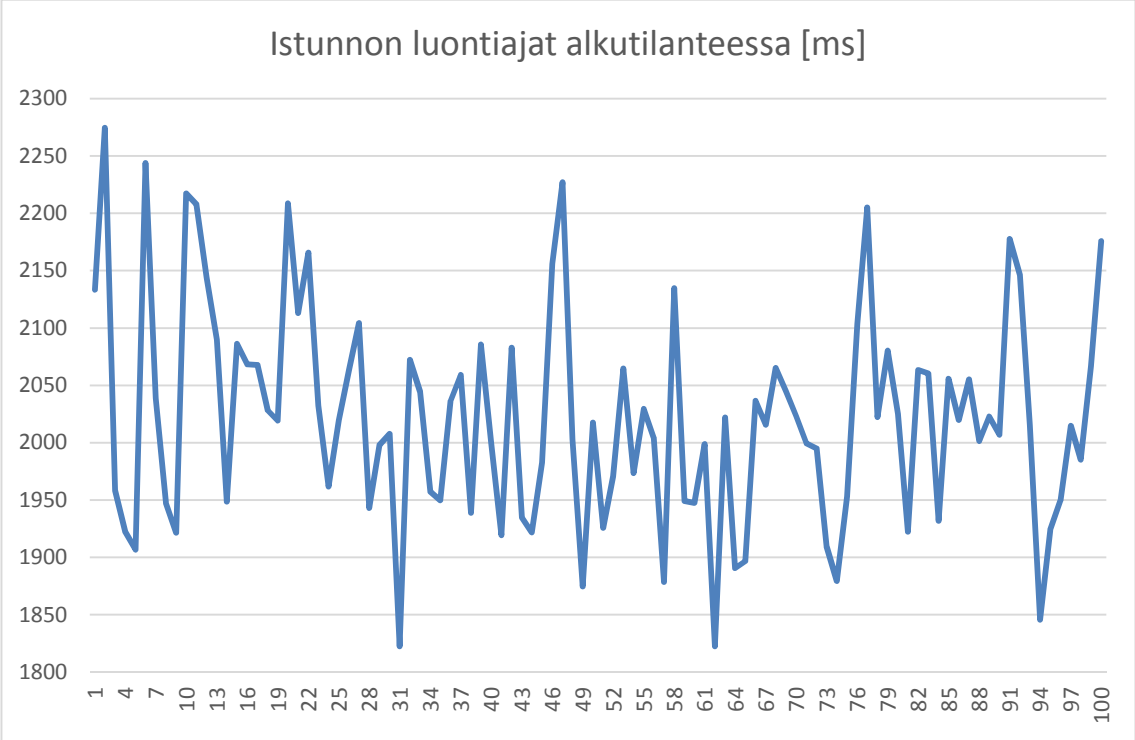
Kuva 1. *ApacheBenchin antamat alkutilanteen tulokset kuvaajana.*

Tarkastellaan vielä kuormitustestin tuloksia kuvaajasta, jotta voidaan arvioida sitä, kuinka luotettavia taulukkoon 3 lasketut tunnusluvut ovat. Kuten kuvasta 1 voidaan todeta, että tuloksissa ei esiinny valtavan suuria eroja, joten esimerkiksi keskiarvo- ja mediaaniaikoja voidaan pitää luotettavina.

2.4 Istunnon luontiajat

ApacheBench-testin yhteydessä Java-koodiin lisättiin tulostukset, joista nähdään istunnon luomiseen kuluva aika nanosekunnin tarkkuudella. Koodiin on jäänyt vanhastaan monia jo osin vanhentuneita piirteitä. Aiemmin joitakin asioita on laskettu käyttäjälle valmiiksi jo istuntoa luodessa, jotta käyttäjän ei ole tarvinnut odotella näitä sivustoa käyttäessään. Nykyisin nämä asiat on kuitenkin jo opittu tekemään paremmin, joten liian suurta viivettä ei näistä synny. Lisäksi käyttöhistorian perusteella on voitu todeta, että yli puolet käyttäjistä ei tarvitse osaa näistä valmiiksi lasketuista ominaisuuksista. Tästä syystä aion samassa yhteydessä siirtää näiden laskennan pois istunnon luontivaiheesta niin, että kaikki nämä tiedot lasketaan vasta sitten, jos ja kun käyttäjä niitä tarvitsee. Kolmas tärkeä

seikka on ForeAmmatin lisääntynyt sosiaalisen median hyödyntäminen. Näistä palveluista tulee paljon sivulatauksia niin sanottujen web-crawlerien toimesta. Koska ForeAmmatin istuntopohjaisuus käyttää hyväkseen evästeitä, joita crawlerit eivät hyödynnä, niin jokainen sivulataus luo uuden istunnon. Päätin tämän työn yhteydessä minimoida myös istunnon luomiseen kuluvan ajan sekä poistaa lisäksi ammatti- ja aluevalintojen koodin muilta kuin ihmiskäyttäjiltä. Varsinainen Java-koodi ei oikeastaan kuulu tämän työn HTML-sivujen sisällöntuottamista käsittelevään osuuteen, mutta koska tällä on vaikutusta myös sivuston ensimmäisen sivun latausaikaan (jolloin istunto luodaan), päätin sisällyttää työhöni myös nämä tulokset.



Kuva 2. *Istunnon luontiajat alkutilanteessa.*

Istunnon luomiseen kuluvat ajat selviävät kuvasta 2. Tämä tulos on tallennettu samasta ApacheBench-kuormitustestistä, joka tehtiin alaluvussa 2.3. Koneellinen testaaminen ei vaikuta tähän, ja selaimella luotujen istuntojen luontiajat ovat myös vastaavat.

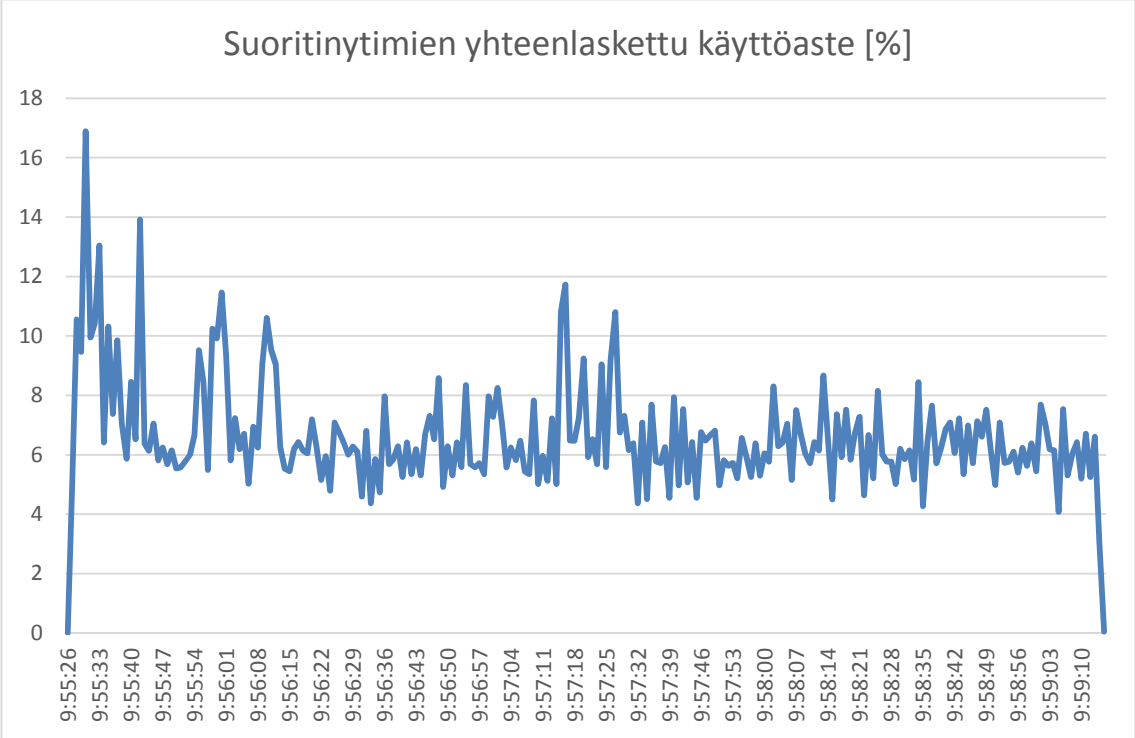
Taulukko 4. *Istunnon luontiaika alkutilanteessa.*

	Minimi	Mediaani	Keskiarvo	Maksimi	Keskihajonta
Istunnon luontiaika	1822 ms	2019 ms	2022 ms	2274 ms	95 ms

Tuloksista lasketaan tunnusluvut taulukkoon 4 ja lopputulokseksi saadaan vaihteluväli 1 822 - 2 274 ms sekä keskiarvoksi 2 022 ms ja keskihajonnaksi 95 ms. Istunnon luontiajat lisäävät siis sivuston ensimmäisen sivun latausaikaa huomattavasti. Tässä olisi hyvä päästä ainakin kymmenesosaluokkaan. Asetetaan tavoitteeksi korkeintaan 200 ms.

2.5 Suoritinytimien kuormittuminen

Suoritinytimien yhteenlaskettua käyttöastetta tarkkaillaan sar-työkalun avulla saman alaluvussa 2.3 suoritetun kuormitustestin aikana. Sar-työkalu kertoo kuormitusasteen jokaiselle suoritinytimelle erikseen, joten luvut on laskettu yhteen kaikkien 24 suoritinytimen osalta tulosten tarkastelua varten.



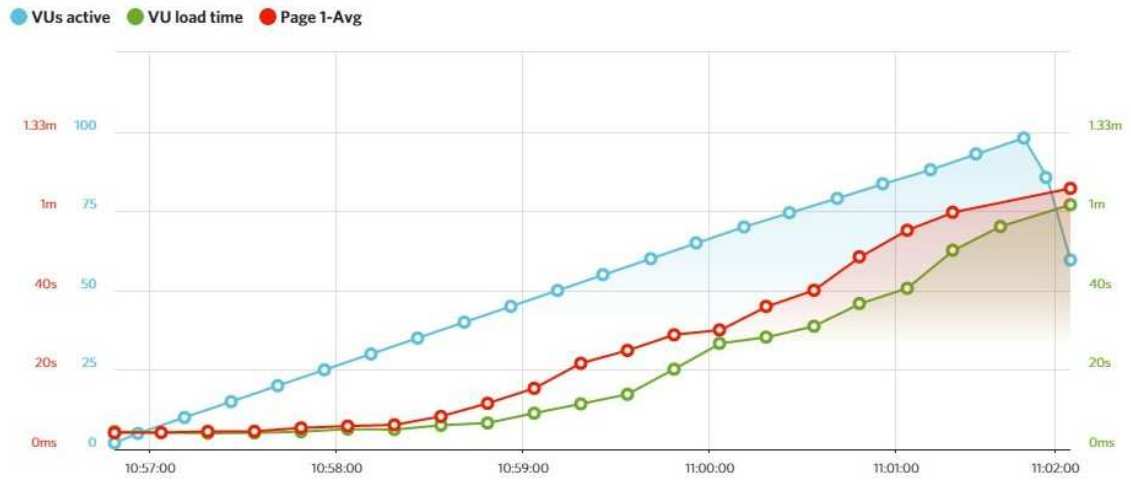
Kuva 3. *Suoritinytimien yhteenlaskettu käyttöaste alkutilanteessa.*

Tarkastellaan suoritinytimien yhteenlaskettua käyttöastetta kuvasta 3. Näytteitä on otettu sekunnin välein testin kestäessä 230 sekuntia. Java-sovelluspalvelimen käynnistämisen jälkeen olen avannut sivun kerran nettiselaimella, jotta Javan virtuaalikoneen prosessi varaa sille tarkoitetun kahden gigatavun muistimäärän. Tämä ei kuitenkaan poista sitä tosiasiaa, että kuormitustestin aloitus saa aikaan heti alussa suorittimien käyttöasteeseen suurimman huippuarvon 16,89 %. Sen jälkeen käyttöaste näyttäisi vakiintuvan melko hyvin eikä suuria piikkejä synny. Testin lopuksi käyttöaste putoaa luonnollisesti nollaan. Koko testijakson mediaaniarvo on 6,25 % ja keskiarvo puolestaan 6,60 %. Keskihajonnaksi muodostuu 1,82 %.

2.6 Load Impact -kuormitustesti

Suurempaa rasitusta simuloidaan Load Impact -kuormitustestillä, jossa simuloitujen käyttäjien määrä kasvaa viiden minuutin aikana sataan yhtäaikaiseen käyttäjään. Tässä testissä yksi käyttäjä tekee useita sivupyyntöjä ja lataa myös sivuun liittyviä resurssitiedostoja. Yhteensä viiden minuutin aikana siirtyy 206,4 megatavua dataa, pyyntöjä tehdään

3 640 kappaletta ja etusivu latautuu 241 kertaa minimiajan ollessa 3,71 sekuntia. Rasitusgeneraattorin sijainti oli testissä Ashburn, Virginia (VA), Yhdysvallat (USA).



Kuva 4. *Yhtäaikaisten käyttäjien kokemat latausajat alkutilanteessa.*

Sivusto piirtää rasitustestistä myös automaattisesti kuvaajan, josta näkyy simuloitujen yhtäaikaisten käyttäjien määrä, käyttäjän kokema viive sekä sivun latausaika. Kuvasta 4 voidaan arvioida, että viive ja latausaika pysyvät vakiona noin 30 käyttäjään asti ja tuplaantuvat 40 ja 50 käyttäjän välimaastossa. Tämä rasitustesti onkin äärimmäinen tapa havainnoida yhtäaikaisten kävijöiden määrää sivustolla, jossa on paljon visuaalista materiaalia sekä informatiivista dataa käyttäjälle luettavaksi. Käytännössä siis sivustoa yhtä aikaa selailevien käyttäjien kuormitus ei koskaan yllä sellaiselle tasolle, jota tämä kuormitustesti simuloi. Asetetaan kuitenkin tavoitteeksi, että kuvaajan punainen ja vihreä viiva pysyvät lopussa liki main vakioina ainakin 50 käyttäjään asti.

2.7 Alkutilanteen yhteenveto

Alkutilanteesta on nyt saatu selville tärkeimmät perusasiat ja samalla on huomattu, että kaikkiaan 29 resurssista koostuva ForeAmmatin etusivu sisältää 14 oman palvelimen ulkopuolelta ladattavaa resurssia. Näistä kymmenen käyttää GZIP-pakkausta, joten sen käyttöönottaminen myös ForeAmmatin palvelimella lienee järkevää. Loput neljä resurssia ovat kaksi kappaletta noin 18 kilotavun kokoisia fontteja sekä pienet Inspectlet- ja Facebook-lisäosien autentikoinnit, joiden hyötykuormat ovat vain 31 ja 43 tavua.

Tarkastellaan vielä itse XHTML-sivun sisältöä ja jaetaan sen sisältö muutamaan osaan. Vertailussa käytetään sitä puhdasta HTML-tiedostoa, jonka käyttäjä palvelimelta saa prosessoinnin päätyttyä. Etusivu sisältää yhteensä 411 140 merkkiä, joista 7 819 merkkiä on skandinaavisia kirjaimia. Skandinaavisten kirjaimien koodaaminen UTF-8-merkistökoodauksella aiheuttaa tiedonsiirtoon enemmän kuin yhden tavun merkkiä kohden. Koska nämä kirjaimet liittyvät erityisesti suomen kieleen eivätkä sivun rakenteeseen, niin voidaan olettaa, ettei niiden määrä juurikaan tule vähentymään koodiin tehtävien muutosten

takia. Siksi käytetäänkin vertailussa selvyyden vuoksi merkkimääriä eikä sotketa asiaan merkistökoodauksia.

- Ammatti- ja aluevalitsimet yhteensä, 373 212 merkkiä ≈ 90,8 %
- josta HTML-koodia sekä tekstiä, 116 885 merkkiä ≈ 31,3 %
- ja JavaScript-koodia, 256 327 merkkiä ≈ 68,7 %
- josta ammattinimikehakua varten. 248 735 merkkiä ≈ 97,0 %

Näin ollen voidaan siis todeta, että ForeAmmatin etusivun XHTML-tiedoston sisällöstä noin 90,8 % muodostuu ammatti- ja aluevalitsimista. Vaikka muilla sivuilla varsinaisen datan määrä onkin etusivua suurempi, niin valitsimien osuus pysynee silti valtavana verrattuna esimerkiksi varsinaisen datan merkkimääriin. Työn päätavoite on siis löydetty. Ammatti- ja aluevalitsimet tulee toteuttaa jollakin muulla tavalla, jotta niiden osuus koko sivusta saadaan paljon pienemmäksi. Valitsimiin liittyvä HTML-koodi tulee pohtia uudelleen ja JavaScriptin puolelta erityiseksi ongelmakohdaksi paljastuu ammattinimikehaku, kuten yllä olevista prosenteista voidaan hyvin todeta. Lopulta jäljelle jäävä JavaScript-koodi tulee mahdollisuuksien mukaan erottaa erilliseen tiedostoon tai useampaan, vaikka se lisääkin yhteismerkkimäärää ja useampien ladattavien tiedostojen myötä myös tiedonsiirtopakettien otsikkotietoja, mutta silloin itse JavaScript-tiedostoille voidaan käyttää selaimen välimuistia.

Sivuston optimoimiseen, tehokkuuden nostamiseen ja latauskuorman pienentämiseen tehokkaimmiksi keinoiksi arvioisin sisällön minimoinnin, GZIP-pakkauksen käyttöönoton sekä selaimen välimuistin hyödyntämisen oikeiden otsikkotietojen avulla. Pakkaaminen tietysti lisää suoritinytimien kuormitusastetta sekä otsikkotietojen kokoa. Palvelimen puolella kuormituksen mittaaminen saattaa jopa onnistua, mutta käyttäjälle esiintyvä lisäkuormitus onkin varmasti vaikeammin arvioitavissa.

3. AMMATTI- JA ALUEVALITSIMIEN OPTIMOINTI

Ammatti- ja aluevalitsimet muodostavat siis 90,8 % ForeAmmatin etusivun merkkimäärästä. Valitsimien sisällä puolestaan ammattinimikehaun JavaScript-koodi muodostaa 97,0 % valitsimien JavaScript-koodista. Oletan, että HTML-osuus olisi helpompi, joten tutkitaan sitä ensin ja JavaScriptiä sen jälkeen. Jaetaan valitsimien optimointi siis kahteen osaan tämän helpohkon luokittelun mukaisesti.

3.1 Valitsimien HTML-koodin optimointi

HTML-koodi muodostetaan siis dynaamisesti sen perusteella, minkälainen tulosjoukko haetaan tietokannasta. Tutustutaan hieman tarkemmin esimerkiksi ammattien listaamiseen. Lähtökohtana on tietysti tietokanta, jossa oleva taulu sisältää virallisen ammattiluokituksen [2] kokonaisuudessaan. Kaikista ammateista löytyy yksilöllinen ID-arvo sekä ammatin virallinen nimi. Sen lisäksi on sarake, joka kertoo, mitkä 200 ammattia kaikista Suomessa käytössä olevista 411 ammatista löytyvät ForeAmmatin ilmaisversiosta. Olemme joutuneet valitsemaan sopivan joukon suurimpia ja merkittävimpiä ammatteja, joista pystymme tarjoamaan kaiken mahdollisen datan. Pienimmissä ammateissa voi olla niin vähän työllisiä, että käyttämämme työmäärä suhteessa käyttäjien saamaan hyötyyn olisi aivan liian suuri.

Java-koodi sisältää siis metodin, joka suorittaa tietokantaan kyselyn, jolla äsken mainitut tiedot saadaan tulostettua. Tämän metodin paluuarvo on tietokannan tulosjoukko, joka voidaan suoraan kiinnittää HTML:ssä dynaamiseen elementtiin, joka näkyy loppukäyttäjälle tavallisena HTML-taulukkona table-elementin sisällä. Tämä on hyvin yksinkertainen ja nopea tapa tulostaa dataa tietokannasta. Koko taulukon rakentamiseen vaadittava merkkimäärä on vain noin 735 merkkiä per ammattiluokituksen pääryhmä, joita on kymmenen kappaletta. Yhteensä siis noin 7 350 merkkiä. Tämä sisältää jo taulukon, sen rivien ja sarakkeiden tyyliluokat sekä kiinnitykset Java-koodiin sekä taulukon tulostamista että ammattinimen klikkaamista varten.

```
<tr class="paritonrivi">
<td><a href="#" onclick="mojarra.jsfcljs(document.getElementById('lomake'),
      {'lomake:j_idt154:4:j_idt156':'lomake:j_idt154:4:j_idt156'},'');return false"
      class="taso2_rivi amlt amlt4">Toimitusjohtajat ja pääjohtajat</a>
</td>
</tr>
```

Kuva 5. *Yhden ammatin rivi taulukossa alkutilanteessa.*

Tällä tulostustavalla saatu HTML-muotoinen lopputulos on kuitenkin kaikkea muuta kuin kompakti (kuva 5). Jokaiseen ammattinimeen liittyy JSF-tekniikkaan (JavaServer Faces)

viittaava JavaScript-koodi, jonka avulla vain JSF tietää sisäisesti mitä tulee tehdä, kun ammatin nimeä klikataan. Tässä tapauksessa tietysti klikkauksesta parametrina saatava ammatin ID-arvo tulee asettaa oikeaan muuttujaan ja sen perusteella hakea sivustolle uudet tiedot siitä ammatista, jota käyttäjä halusi katsella. Tämä lopputulos on kuitenkin ilman ammatin nimeäkin keskimäärin 222 merkkiä pitkä. Ammattien nimiä on monenlaisia ja pituudet vaihtelevat viidestä merkistä 64 merkkiin, mutta keskiarvo on kuitenkin noin 30 merkkiä.

Näiden lisäksi taulukoiden aloitus- ja lopetustageista syntyy vielä 30 merkkiä per taulukko. Kun siis lasketaan yhteismerkkimäärää, voidaan käyttää keskiarvona 252 merkkiä jokaiselle 200 ammatille, joiden lisäksi kymmenestä taulukosta tulee vielä 300 merkkiä. Yhteensä tämä tekee siis 50 700 merkkiä. Koska tätä dynaamista XHTML-sivua ei voi tallentaa selaimen välimuistiin, ladataan nämä merkit joka kerta uudelleen, jokaisella sivulla, vaikka sisältö ei muutukaan.

3.1.1 Ammattitaulukoiden välimuistitus

Ensimmäinen ajatus on siis erottaa tämä kiinteän tekstin kaltainen osuus omaan tiedostoonsa, jonka tiedostopääte on jokin muu kuin XHTML-sivujen tiedostopääte. Tässä tapauksessa eri tiedostopäätteisen tiedoston voisi tallentaa käyttäjän selaimen välimuistiin. Ulkoisen tiedoston sisällön liittäminen mihin tahansa kohtaan sivua on myös helposti toteutettavissa yhdellä komennolla. Tässä kuitenkin törmättiin melko pian siihen tosiasiaan, että JSF ei osaa käsitellä JSF:n sisäisiksi tarkoitettuja linkkejä, ellei JSF:n taustalla pyörivä Faces Servlet käsittele näitä sivuja. Tämä käsitteleminen taas puolestaan vaatii sitä, että sivu on XHTML-sivu, jolloin sivua ei myöskään tallenneta selaimen välimuistiin. Tämä vaihtoehto piti siis hylätä toimimattomana.

3.1.2 Ammattilinkkien muuttaminen ulkoiseksi

Oletusarvoisesti käyttäjän selain välittää POST-paketilla tiedot käyttäjän tekemisistä palvelimelle, jossa JSF tietää, miten näitä tulee käsitellä. Olin kuitenkin hieman aiemmin saanut kehitettyä Java-koodiin käsittelijän, joka lisää URL-osoitteen (Uniform Resource Locator, verkkosivun osoite) perään ammatin ja alueen kyselyparametreina. Alun perin tämä oli tarkoitettu käytettäväksi suoriin linkkeihin esimerkiksi kirjanmerkeissä tai sosiaalisessa mediassa. Kyselyparametrien avulla voitiin siis luoda toimiva linkki jollekin ForeAmmatin sivulle myös niin, että valitaan valmiiksi tietty ammatti ja/tai alue, jolloin käyttäjän istunto ei aina alakaan tyhjillä tiedoilla.

Tämä koodi on kuitenkin suhteellisen monimutkainen, vaikka rivejä siinä onkin vain noin 115, mutta erilaisia vaihtoehtoja on monia, koska osoitteen pitää päivittyä, jos käytetään sivuston valitsimia. Jos taasen kirjoitetaan kyselyparametrit osoitteeseen, pitää tietojen ja valitsimien päivittyä vastaamaan niitä. Tätä koodia en käsittele tässä työssä sen enempää, mutta se oli kuitenkin näiden muutosten esivaatimus, jotta ammatti- ja aluevalitsimien

optimointi oli ylipäätään mahdollista. Tätä tulevaa käyttöä ulkoisille linkeille en tiennyt vielä silloin, kun kyseistä koodia kirjoitin.

```
<tr class="paritonrivi">
<td><a href="/index?ammatti=1120"
      class="taso2_rivi amlt amlt4">Toimitusjohtajat ja pääjohtajat</a>
</td>
</tr>
```

Kuva 6. *Yhden ammatin rivi taulukossa ulkoisella linkillä.*

Nyt siis JSF:n sisäinen linkki voidaankin korvata ulkoisella linkillä (kuva 6), joka viittaa samaan sivuun, jolla käyttäjä tällä hetkellä on, mutta jonka perään vain sijoitetaan kyselyparametrina uusi ammatti. Etusivun tapauksessa tämä tarkoittaisi 138 merkkisen JSF:n sisäisen linkin korvaamista 29 merkkisellä ulkoisella linkillä. Säästöä syntyisi 200 ammatin tapauksessa yhteensä näin ollen 21 800 merkkiä, jolloin jäljelle jäisi vielä 28 900 merkkiä. Merkkimäärä pienentyisi pelkästään ulkoisten linkkien avulla siis noin 43 %. Tämän muutoksen toteuttamisen jälkeen jatketaan kuitenkin vielä vaihtoehtojen perusteellisempaa tutkimista.

3.1.3 Ammattilistojen luominen dynaamisesti

Jäljellä on siis vielä keskimäärin 143 merkkiä jokaiselle 200 ammatille sekä 300 merkkiä taulukoiden aloitus- ja lopetustageja. Yksittäisten ammattien rivien tarkempi analysointi paljastaa, että taulukon HTML-koodia on 18 merkkiä ja taulukkotyylejä keskimäärin 26,5 merkkiä, jotka pystytään poistamaan, jos listat muodostetaan ilman taulukkoja. Koko listauksessa nämä ”ylimääräiset” osat tekevät siis yhteensä vielä 9 200 merkkiä. Ilman nykyistä kutsukoodia listojen dynaamisuus ja silmukkarakenteet pitää kuitenkin luoda itse. Säästö näyttää kuitenkin sen verran lupaavalta, että päätän siihen ryhtyä.

```
var ammt4_ID = [1111,1112,1114,1120,...];
var ammt4_nimi = ["Lainsäätäjät","Julkishallinnon ylimmät virkamiehet",
      "Järjestöjen johtajat","Toimitusjohtajat ja pääjohtajat",...];
```

Kuva 7. *Ammattiluokituksen tietoja JavaScript-muuttujissa.*

Luodaan JavaScript-tiedosto, johon perustetaan vektorit sekä ammattien ID-tunnisteille että ammattien nimille. Liitetään nämä tiedot niin sanotusti kovakoodattuna tietokannasta JavaScript-vektoreihin (kuva 7). Nämä tiedot ovat 16 132 merkkiä pitkiä, mutta näihin pystytään myöhemmin käyttämään selaimen välimuistia ja toki GZIP-pakkausta myös. Ne eivät myöskään käytännössä muutu, korkeintaan pari-kolme kertaa vuodessa, joten näin voidaan tehdä. Sen jälkeen muutetaan Java-koodin tietokantakyselyitä niin, että kyselyn ja silmukkarakenteen jälkeen paluuarvo onkin vain haluttujen ammattien ID:t oikeassa järjestyksessä merkkijonona.

```
function write_ammt4(ID,type) {
  ...  for (var i=0; i<ammt4_ID.length; i++) { if (ammt4_ID[i] == ID) {
            if (type == 'div' && ...) { document.write(...); }
  ...  } } }
var lkpr_luokitus_jono1 = [1112,1114,1120,...];
for (var i=0; i<lkpr_luokitus_jono1.length; i++) {
      write_ammt4(lkpr_luokitus_jono1[i],'div'); }
```

Kuva 8. *Tarvittavien ammattilinkkien muodostaminen luokituksen perusteella.*

Lopuksi rakennetaan hieman JavaScript-koodia, joka saatujen merkkijonojen perusteella käy pääryhmä kerrallaan läpi ammatit vertaillen ID-arvoja ja sopivan osuessa kohdalle, kirjoittaa siihen kohtaan sivua tavallisen div-elementin ja sen sisään linkin käyttäen alaluvussa 3.1.2 kuvattua ulkoista linkkiä ja linkin näkyväksi tekstiksi ammatin nimen (kuva 8). Näin jäljelle jää noin 2 000 merkkiä silmukoita suorittavaa koodia ja noin 2 000 merkkiä silmukoita kutsuvaa koodia. Ammattilistojen muodostaminen on annettu käyttäjän selaimen tehtäväksi ja XHTML-sivusta, jota ei voi tallentaa välimuistiin, on saatu poistettua noin 110 000 merkkiä. Tämä uusi JavaScript-koodi suoriutuu tehtävästään muutamassa millisekunnissa, joten laskentakuorma ei oleellisesti lisäänny.

3.2 Valitsimien JavaScript-koodin optimointi

Ammattinimikehaun ensimmäinen versio on toteutettu siten, että erilliseen tietokannan tauluun päivitetään haun vaihtoehdot automaattisesti kerran viikossa. Tähän poimitaan mukaan sekä ammattiluokituksen [2] virallisia ammattien nimiä että Tilastokeskuksen listaamia ammattinimikkeitä. Esimerkiksi ammatille talousjohtajat Tilastokeskus määrittelee siihen kuuluvaksi ammattinimikkeet talousjohtaja, rahoitusjohtaja ja hallintojohtaja. Vastaavasti kerrotaan, että ammattinimikkeet pankinjohtaja, laskentapäällikkö ja hallintopalvelujohtaja eivät kuulu talousjohtajien ammattiin vaan johonkin muuhun [37]. Lisäksi hakuun tulee mukaan sellaisia ammattinimikkeitä, jotka ovat esiintyneet vähintään kolmessa työpaikkailmoituksessa. Ammattiluokituksen viralliset ammattien nimet eivät välttämättä ole kovinkaan kuvaavia, joten tämän työn yhteydessä ammattinimikkeellä tarkoitan termiä, jonka esimerkiksi mahdollinen työnhakija ymmärtäisi paremmin. Hyvänä esimerkkinä vaikkapa ammattinimike hovimestari kuuluu nykyään ammatin ravintola- ja suurtaloustyöntekijät alle (5120) [33]. Vanhassa ammattiluokituksessa, jonka käytöstä luovuttiin Suomessa kesällä 2014, hovimestareillekin oli oma ammattinsa [1].

Istuntoa luodessa tämä taulu luetaan läpi ja sen ID-arvoista, ammattinimikkeistä sekä ammattien virallisista nimistä pyöritetään silmukkarakenteella merkkijonot, jossa nämä kaikki ovat peräkkäin. Näiden merkkijonomuuttujien arvot sijoitetaan JavaScript-muuttujiin sivustolla. Tästä johtuen merkkimäärää kertyy huomattavan paljon, koska ammattinimikkeitä tulee koko ajan lisää. Työpaikkailmoituksia jättävät ovat nyt oppineet käyttämään uutta ammattiluokitusta paremmin ja sen tarkinta kansallista tasoa on myös täydennetty jo useita kertoja vastaamaan Suomen tarpeita.

Lisäksi HTML-koodissa oli vastaava taulukkorakenne kuin aiemmin kuvatuissa ammateissakin, joka sisälsi hakutuloksia varten kaikki ammatit. Käyttäjän kirjoittaessa kirjaimia hakukenttään, etsittiin JavaScript-muuttujan sisältämistä ammattinimikkeiden merkkijonoista vastaavuuksia ja ammattien virallisten nimien perusteella piilotettiin tai näytettiin taulukosta löytyvät ammatit. Haku tapahtui siis kokonaan käyttäjän koneella ja sen nopeus riippui tietysti sekä koneen suorituskyvystä että selaimesta itsestään.

Ammattinimikehaun ensimmäistä versiota tehdessäni törmäsin kahteen suureen ongelmaan, joista johtuen hausta tuli näinkin monimutkainen ja siirtokapasiteettia kuluttava. Ensinnäkään tuloksia ei pystynyt kirjoittamaan JavaScriptillä JSF:n sisäisinä linkkeinä, vaan linkkien tuli olla olemassa jo sivua luotaessa. Tällöin ei siis ulkoisten linkkien käsittelyä vielä ollut koodattuna. Toisekseen taulukon sisään ei pystynyt asettamaan elementeille ID-arvoja tietokannan tulosjoukon perusteella, esimerkiksi ammatin ID-arvoa. Tästä syystä piti käyttää vielä virallisia ammattinimiä taulukon rivien piilottamiseen ja näyttämiseen.

Suorittamiseen kuluvan ajan mittaan sekä alussa että lopussa saman koneen samalla selaimella kirjoittaen samat kirjaimet. Tulos ei silti tule olemaan täysin luotettava, koska koneeni on jo aiemmin mainittu perinteinen Windows-kone, joten erilaiset taustalla ajettavat prosessit saattavat hieman vaikuttaa suorituskykyyn. Haku kuitenkin kestää silmämääräisestikin pidempään, jos sanaa löytyy ammattinimikkeistä paljon. Esimerkiksi kirjaimilla ”ups” löytyy muun muassa upseereihin liittyviä tuloksia haun kestäessä noin 2 - 4 ms. Kuitenkin kirjaimilla ”myy” löytyy monia ammatteja, koska myyntiin liittyviä nimikkeitä on myös muissa kuin varsinaisissa myyntityöammateissa. Tämä haku kestää noin 125 - 135 ms.

3.2.1 Ammattinimikkeiden haku palvelimelta

Olin jo aiemmin useaan otteeseen lukenut AJAX-tekniikasta (Asynchronous JavaScript and XML), jolla voisi esimerkiksi päivittää pienen osan sivustoa asynkronisesti. Tällöin käyttäjän selain lähettäisi pienen paketin palvelimelle, joka käsittelisi saamaansa tietoa jotenkin ja palauttaisi vastauksena pienen paketin. Näiden tietojen perusteella sivun sisältöä voisi päivittää käyttämällä JavaScriptiä ja samalla sivu ei kokonaisuudessaan latautuisi uudelleen. Aluksi en ymmärtänyt, miten JSF ja AJAX liitetään toisiinsa tällaisessa vaativammassa käytössä. Juuri ennen tämän työn aloittamista kuitenkin keksin muussa yhteydessä sen, minkälainen Java-koodi tulee kirjoittaa käsittelemään AJAX-pyyntöjä, jotta useamman elementin perusteella voidaan myös muuttaa useampaa elementtiä. Tämän oivaltamisen jälkeen ajatukset siirtyivätkin suoraan ammattinimikehakuun.

```
<div id="valinnat1_ammth" class="taso1_rivi">
      <span id="haku_teksti" class="hakuboxi">Haku:</span>
      <input id="lomake:hakuboxi1" type="text" autocomplete="off" value=""
      onfocus="mojarra.ab(this,event,'focus','lomake:hakuboxi1','lomake:nimikeResults1'"
      onkeyup="mojarra.ab(this,event,'keyup','lomake:hakuboxi1','lomake:nimikeResults1'"
      />
</div>
<div class="nimikeResults">
      <input id="lomake:nimikeResults1" type="hidden" value="" />
</div>
```

Kuva 9. *Uudet AJAX-tekniikkaa käyttävät ammattinimikehakulaatikot.*

Tarkoituksena on nyt muuttaa Java-koodia niin, että ammattinimikkeet luetaan tietokannasta istunnon luonnin yhteydessä ja jätetään tämä tietokannan tulosjoukko sellaisenaan palvelimen muistiin. Muutetaan sivuston ammattinimikehaun syötekenttää niin, että se lähettää arvonsa AJAX-pyyntönä palvelimelle arvon muuttuessa. Lisätään sivustolle myös haun tuloskenttä, johon AJAX-vastaus sitten kirjoittaa niiden ammattien ID-arvot, jotka vastaavat hakua (kuva 9). Haun käsittelemiseksi luodaan palvelimelle koodi, joka vertaa hakuarvoa em. tulosjoukkoon poimien siitä oikeat ID-arvot. Nykyisellä osaamisellani tämä olikin nyt melko helposti toteutettavissa ja tehokkaalta palvelimelta tulosten etsiminen kesti kirjaimilla ”ups” noin 1 - 2 ms ja kirjaimilla ”myy” noin 7 - 8 ms.

3.2.2 Hakutulosten muuttaminen näkyviksi

Seuraavaksi haun tuloskentän arvojen perusteella pitäisi suorittaa silmukka, joka etsii nämä ammattien ID-arvot JavaScript-muuttujasta, joka luotiin aiemmin ammattilistojen dynaamisen luonnin yhteydessä. Tulosten perusteella voidaan luoda hakutuloksista linkkilista, johon ammatin ID-arvo sisällytetään aiemmin luodun ulkoisen linkin periaatteella ja virallinen nimi poimitaan toisesta muuttujasta. Ongelmaksi kuitenkin muodostui se, ettei tähän piilotettuun tuloskenttään voinut kiinnittää minkäänlaisia tapahtumankäsittelijöitä. Asia ratkesi löydettyäni BalusC:n Stack Overflow -yhteisön sivuille kirjoittaman ohjeen [14], jonka mukaan JSF:n yhteydessä syötekenttään voidaan kiinnittää yleinen AJAX-tapahtumia valvova tapahtumankäsittelijä, joka suoritetaan yhteensä kolme kertaa. Ensimmäisen kerran ennen kuin AJAX-pyyntö lähetetään, toisen kerran AJAX-vastauksen vastaanottamisen jälkeen ja kolmannen kerran, kun AJAX-vastauksen määräämät kentät on päivitetty JavaScriptin avulla. Näistä viimeinen oli se, jota nyt tarvitsin siihen, että ammattien lista muodostetaan hakutulosten perusteella. Ensimmäistä puolestaan voin käyttää siihen, että käynnistän ajanoton, kun mittaan koko prosessin suoritusaikaa.

```
<div id="valinnat1_ammth" class="taso1_rivi">
      <span id="haku_teksti" class="hakuboxi">Haku:</span>
      <input id="lomake:hakuboxi1" type="text" autocomplete="off" value=""
      onfocus="mojarra.ab(this,event,'focus','lomake:hakuboxi1','lomake:nimikeResults1',
            {'onevent':tutkiNimike})"
      onkeyup="mojarra.ab(this,event,'keyup','lomake:hakuboxi1','lomake:nimikeResults1',
            {'onevent':tutkiNimike})" />
</div>
<div class="nimikeResults">
      <input id="lomake:nimikeResults1" type="hidden" value="" />
</div>
```

Kuva 10. *Ammattinimikehakulaatikkoon lisätyt onevent-attribuutit.*

AJAX-tapahtumia valvovan tapahtumankäsittelijän avulla (kuva 10) pystyin suorittamaan ammatteja etsivän ja dynaamisesti kirjoittavan silmukkarakenteen sen jälkeen, kun hakutulokset sisältävä piilotettu kenttä on päivittynyt. Näin tehden käyttäjälle ei tarvitse etukäteen ladata sen paremmin ammattinimikkeiden tai ammattien listoja kuin valmista taulukkorakennettakaan. Säästetään siis huomattava määrä merkkejä, joita ei olisi pystynyt tallentamaan käyttäjien selaimien välimuistiin. Huonona puolena tämä tietysti lisää verkkoliikennettä, mutta onneksi pakettien koot eivät ole hirveän suuria nykyisille nettiyhteysnopeuksille. Jokainen paketti kuitenkin sisältää myös istunnon yksilöivää tunnistetietoa, jota en pysty nykyisen tekniikan avulla vähentämään. AJAX-pyyntöjen pakettikoot otsikkotietoineen vaihtelevat välillä 4 400 - 4 900 tavua. AJAX-vastausten pakettikoot puolestaan vaihtelevat otsikkotietoineen välillä 2 700 - 2 900 tavua. Yhteensä siis pahin mahdollinen yhdistelmä yltää kokoon noin 7 800 tavua. Mikäli käyttäjä kirjoittaa esimerkiksi viisi kirjainta, tulee tästä yhteensä noin 39 kilotavua.

3.2.3 Ammattinimikehaun kesto

Kun mittaan hakuprosessin kestoa kirjaimilla ”myy”, kestää yleensä tapahtumankäsittelijän ensimmäisen ja toisen vaiheen välissä noin 80 - 90 ms ja toisen ja kolmannen vaiheen välissä noin 8 - 11 ms. Palvelimen suorittaman haun kesto sisältyy siis ensin mainittuun aikaan. Näin ollen lopputuloksena voidaan todeta, että AJAX-tiedonsiirto kestää noin 72 - 82 ms, palvelimen suorittama haku noin 7 - 8 ms, tulosten päivittäminen noin 8 - 11 ms ja koko hakuprosessin yhteiskesto vaihtelee välillä 85 - 95 ms. Seuraavaksi kellotetaan hakuprosessi kirjaimilla ”ups”. Tapahtumankäsittelijän ensimmäisen ja toisen vaiheen välissä kestää noin 70 - 80 ms ja toisen ja kolmannen vaiheen välissä noin 1 - 3 ms. AJAX-tiedonsiirto kestää siis noin 68 - 78 ms, palvelimen suorittama haku noin 1 - 2 ms, tulosten päivittäminen noin 1 - 3 ms ja koko hakuprosessin yhteiskesto vaihtelee välillä 70 - 83 ms.

Tiedonsiirtovaiheen kesto vaihtelee millisekuntiluokassa tarkasteltuna hyvin suuresti, eikä tulosten tarkkuus ole kovinkaan suuri, mutta hyvin suuntaa antava kuitenkin. Ajassa tarkasteltuna siis yleisemmin esiintyvien kirjainten ”myy” haku kestää nyt keskimäärin

noin 90 ms entisen noin 130 ms sijaan. Harvemmin esiintyvien kirjainten ”ups” haku kestää nyt noin 76 ms entisen noin 3 ms sijaan. Voidaan siis todeta, että vain vähän osumia sisältävällä hakusanalla tiedonsiirtoon kuluva aika lisää merkittävästi haun kestoa, mutta vastaavasti taas paljon osumia sisältävällä hakusanalla palvelimen suorittama haku on niin paljon tehokkaampaa, että yhteisaika pienenee noin kolmanneksella tiedonsiirrosta huolimatta. Hakuajan pidentyminen vähän osumia sisältävillä hakuehdoilla on kuitenkin pienempi paha, joka pitää nyt hyväksyä, jotta merkkimäärää saadaan pienennettyä.

3.3 Valitsimien merkkimäärän pienentyminen

HTML-tiedoston sisältämä merkkimäärä oli toinen kahdesta asiasta, joita ensisijaisesti lähdettiin pienentämään, koska tätä dataa ei voinut tallentaa selaimien välimuisteihin. Alussa mainittu ammatti- ja aluevalitsimien sisältämä HTML-koodi sekä teksti sisälsivät siis suurimmaksi osaksi ammattilistojen taulukkorakenteet sekä niissä näkyvät ammattien nimet. Aluksi merkkejä oli 116 885 kpl, joista ulkoisilla linkeillä säästettiin aluksi noin 21 800 merkkiä, mutta listojen dynaamiseen luomiseen siirtymisen jälkeen lopullinen merkkimäärä on vain 8 318 kpl. Vähennystä siis 108 567 merkkiä eli noin 92,9 %.

Ammattinimikehakua varten tarkoitettu JavaScript-koodi oli puolestaan toinen merkittävimmistä asioista, joihin oli tarkoitus etsiä vaihtoehtoja. Aluksi kyseinen koodi oli noin 97,0 % kaikesta ammatti- ja aluevalitsimien sisältämästä JavaScript-koodista ja oli pituudeltaan 248 735 merkkiä. Nyt jäljelle jää vain AJAX-tapahtumia valvovan tapahtumankäsittelijän koodi, onnistuneen AJAX-päivityksen jälkeen suoritettava koodi sekä ammattilinkit listaan kirjoittava koodi, ja merkkimäärä on vain 1 642 merkkiä. Vähennystä siis 247 093 merkkiä eli noin 99,3 %.

Valitsimien sisältämää JavaScript-koodia oli alussa kaiken kaikkiaan 256 327 merkkiä ja nyt 8 976 merkkiä. Kun yhteismääristä vähennetään ammattinimikehakua varten tarkoitettu JavaScript-koodi, niin nähdään, että muuta kuin ammattinimikehakua varten tarkoitettua JavaScript-koodia oli alussa 7 592 merkkiä ja nyt 7 334 merkkiä. Vähennystä siis 258 merkkiä eli noin 3,4 %. Melko hyvä tulos siihen nähden, että kuitenkin joukkoon tuli myös uutta JavaScript-koodia, kuten ammattilinkkien kirjoittamista pyytävää koodia, ammattilinkkejä kirjoittavaa koodia sekä tietysti aluevalitsimen osalta samanlaiset koodit.

Kuten aiemmin todettiin, että JSF käsittelee vain alun perin XHTML-päätteiset tiedostot, joten sellainen JavaScript-koodi, joka vaatii kytkentää myös Java-koodin muuttujiin, tulee edelleen sijoitettavaksi XHTML-sivulle selaimien välimuistien ulottumattomiin. Ammattilinkkeihin tarvitaan esimerkiksi kansion nimi sekä yksittäisen sivun nimi, jotka saadaan Java-muuttujista. Varsinaisen ammattiluokituksen, sekä ne funktiot, jotka eivät tarvitse Java-muuttujia, sijoitan erilliseen JavaScript-tiedostoon, joka voidaan tallentaa selaimien välimuisteihin. Tähän tutustutaan hieman myöhemmin alaluvussa 6.3. Tämän tiedoston lataamista myös kutsutaan vasta valitsimien jäsentämisen alkuvaiheessa. Tarkoituksena on, että vain ihmiskäyttäjien selaimet lataavat tämän tiedoston. Tähänkin asiaan

tutustutaan myöhemmin. Tämän ulkoisen JavaScript-tiedoston kooksi muodostuu 31 472 tavua, ja se sisältää 30 622 merkkiä.

Kokonaisuudessaan ammatti- ja aluevalitsimien merkkimäärä oli aluksi 373 212 merkkiä, joka oli 90,8 % koko etusivun merkkimäärästä. Lopuksi valitsimia varten jäi yhteensä 17 294 merkkiä. Vähennystä siis 355 918 merkkiä eli noin 95,4 %. Koko etusivun yhteismerkkimäärä oli aluksi 411 140 merkkiä ja tässä vaiheessa työtä etusivun yhteismerkkimääräksi muodostuu 55 222 merkkiä. Yhteismerkkimäärä on siis vähentynyt samaiset 355 918 merkkiä eli noin 86,6 %. Valitsimien osuus koko sivun merkkimäärästä on nyt siis noin 31,3 % eli vähennystä 59,5 %-yksikköä.

Tässä vaiheessa voidaan siis todeta, että valitsimien optimoinnilla on saavutettu merkittäviä edistysaskelia merkkimäärissä mitattuna. Suorituskykymuutoksia tullaan mittaamaan työn lopuksi. Tässä vaiheessa esitänkin valitsimien osalta yhteenvetotaulukon.

• Ammatti- ja aluevalitsimet yhteensä,	17 294 merkkiä	≈ 31,3 %
• josta HTML-koodia sekä tekstiä, ja	8 318 merkkiä	≈ 48,1 %
• JavaScript-koodia,	8 976 merkkiä	≈ 51,9 %
• josta ammattinimikehakua varten.	1 642 merkkiä	≈ 18,3 %

4. ULKOISTEN RESURSSIEN OPTIMOINTI

Ulkoisilla resursseilla tarkoitetaan esimerkiksi CSS-tyylitiedostoja, JavaScript-kooditiedostoja, kuvia, fontteja sekä liitännäisiä, jotka yleensä sisältävät lisää latauskoodeja, jotka puolestaan lataavat muilta palvelimilta jopa kaikkia edellä mainittuja. Tässä kohtaa ensisijaisena tarkoituksena on optimoida muun muassa resurssien sisältö sekä lataus- ja suoritusjärjestykset. Pakkauksiin ja välimuisteihin tutustutaan myöhemmin luvussa 6.

Tutkitaan ensin selkeimpiä resursseja eli tyylitiedostoja ja JavaScript-tiedostoja. Oletus on, että näiden merkkimäärää pystytään reilusti pudottamaan minimoinnin avulla, koska kummatkin on kirjoitettu mahdollisimman selkeälukuiseksi käyttäen runsaasti välilyöntejä, rivinvaihtoja ja sarkaimia. Samassa yhteydessä kannattaa myös lisätä tiedostoihin kommentteja, koska koneellinen minimointiprosessi poistaa ne joka tapauksessa.

4.1 CSS-tyylitiedostot ja fonttikirjastot

ForeAmmatti käyttää Roboto Slab -fonttia, joka ei oletuksena sisälly selaimiin. Fontti ladataankin Googlen fonttikirjastosta, joka tarjoaa fontit mahdollisimman uudessa ja tehokkaassa tiedostomuodossa, joka toimii käyttäjien selaimissa. Näistä eri fonttimuodoista esimerkiksi WOFF (Web Open Font Format) toimii Firefoxissa, Chromessa ja Internet Explorerissa versiosta 9 alkaen. WOFF on jo valmiiksi pakattu TrueType / OpenType -fontti [45]. Uusin WOFF 2.0 tarjoaa entistä tehokkaamman pakkauksen ja tätä kirjoittaessani selaintuki löytyy työpöytäselaimista Firefoxista, Chromesta ja Operasta. Mobiiliselaimissa tuettuina ovat Android Browser, Opera Mobile, Chrome for Android sekä Firefox for Android. Merkittävimmistä selaimista tuki puuttuu vain Internet Explorerista, Edgestä ja Safarista [46].

Fonttitiedostot ladataan oletuksena vain Latin-merkistöllä, joka sisältää esimerkiksi suomen kielen kannalta oleellisimmat Unicoden kaksi ensimmäistä lohkoa (koodipisteet U+0000 - U+00FF). Ensimmäisestä lohkosta löytyvät perinteiset ASCII-merkit ja toisesta esimerkiksi skandinaaviset kirjaimet. Roboto Slab -fontti on tarjolla neljällä eri paksuudella, mutta näistä käytimmekin jo valmiiksi vain kahta, normaalia ja lihavoitua. Tälle resurssille ei siis tarvitse tehdä nyt mitään, vaan se on jo riittävästi optimoitu. Fonttitiedostoja ei kannata lähteä siirtämään omalle palvelimelle, koska selaintuki on varsin vaihteleva, joten fontit pitäisi tallentaa monessa eri muodossa.

4.1.1 Ylimääräiset tyylit

ForeAmmatin omat tyylimääritykset ovat kaikki yhdessä CSS-tiedostossa. Tyylien tai tyyliluokkien käyttöön ei tässä yhteydessä tullut isoja muutoksia, vain muutamia lisäyksiä ja täsmennyksiä sinne tänne. Tässä vaiheessa keskityttiin vain tiedostokoon minimointiin.

Sivusto käytiin läpi Firefoxin Firebug-liitännäisen laajennusosalla CSS Usage, jolla pystyy kartoittamaan CSS-tiedostosta ylimääräiset rivit. Tällä työkalulla voi skannata vain yhden sivun kerrallaan tai jättää se automaattiskannaukselle. Lopuksi työkalu kertoo tuloksissaan, moneenko elementtiin kukin tyyli vaikutti.

Laitetaan automaattiskannaus päälle ja käydään selaimella läpi kaikki sivut. Tuloksissa näkyy heti punaisella ne rivit, joiden osumamäärä jää nollaksi. Tuloksia voi käydä läpi joko käsin tai tallentaa suoraan työkalusta käsin uuden version, josta nämä ylimääräiset tyylit on poistettu. Dynaamisella sivustolla on kuitenkin vaikea käydä kaikkia vaihtoehtoja läpi, koska myös Java-koodin muuttujiin on kiinnitetty sivujen renderöintiin vaikuttavia ehtoja. Helpoin esimerkki on esimerkiksi kirjautuminen. Kirjautuneena näytetään uloskirjautumisvaihtoehdot ja muuten sisäänkirjautumisvaihtoehdot. Joitain kohtia jäi itseltänikin huomaamatta, joten suosittelen tyylien läpikäymistä käsin. Aluksi on järkevää vaikkapa kommentoida nämä ylimääräisiksi luokitellut tyylit ja jatkaa selailua. Mikäli kaikki näyttää olevan kunnossa, voi ylimääräiset tyylit poistaa vasta lopuksi.

Samassa yhteydessä on hyvä myös hieman ryhmitellä tyylejä erilaisten kommenttien alle, mikäli tyylien löytäminen tiedostosta on tuntunut vaikealta. Itse päädyin likimain seuraavaan ryhmittelyyn: kaikilla sivuilla tarvittavat selainten oletustyylien ”nollausmääritykset”, kaikkien sivujen rakenteelliset yhteisasetukset, responsiivisuuden määritykset, kirjautumislaatikon tyylit, valitsimissa käytetyt tyylit, mainoksissa käytetyt tyylit, headerissa ja footerissa käytetyt tyylit, etusivun tyylit, ammattisivujen tyylit, infosivujen tyylit, blogisivujen tyylit, vain yhden määrityksen sisältävät yleistyylit sekä muut jokaiselle sivulle erikseen.

4.1.2 Tyylitiedoston minimointi

Tyylitiedoston minimoinnilla tarkoitetaan sellaista koneellista operaatiota, jolla CSS-merkkaus muutetaan selkeälukuisesta muodosta tiiviimpään, mutta silti edelleen toimivaan muotoon [43]. Tyypillisimpiä itse sisältöön vaikuttavia muutoksia ovat esimerkiksi RGB-värikoodien muuntaminen heksadesimaalimuotoon, puolipisteen poistaminen viimeisen määrityksen lopusta sekä fontin painon määrittäminen lukuna tekstin sijaan [8]. Esimerkiksi ”ForeSinisen” (keksimämme termi värille, jota ForeAmmatti käyttää paljon) värin voi esittää RGB-muodossa rgb(79,129,189) tai heksadesimaalimuodossa #4F81BD eli 15 merkin sijasta tarvitaan vain seitsemän merkkiä ja mikäli yhden värikanavan heksadesimaaliesitys sisältää kaksi samaa merkkiä, se voidaan esittää yhtenä, jolloin se osataan automaattisesti tulkita oikein.

Etsin Googlen hakukoneen avulla muun muassa termillä ”crunch” tai ”cruncher”, joka on yleisesti käytetty termi, kun tarkoitetaan minimointia tai tiivistämistä. Kokeilin sattumanvaraisesti yhdeksää erilaista ensimmäisenä löytämääni sivustoa, joista kuusi antoi toimivalta näyttäviä tuloksia. Alkujaan tiedoston koko oli 34 111 tavua ja tulokset vaihtelivat välillä 28 287 - 30 603 tavua. Ensimmäisenä ajattelin kokeilla tietysti pienimmän koon

antanutta CSS Compressor -sivustoa [8], joka osoittautuikin toimivaksi minimoinniksi. Sivuston ulkoasu oli pysynyt ennallaan.

Oletusasetuksilla tiivistämisen taso onkin jo korkein, joka saa aikaan pienimmän tiedostokoon muuttaen CSS-tiedoston sisällön hyvin hankalasti luettavaan muotoon. Oletusasetukset suorittavat niin ikään myös äsken mainitsemani sisällön optimoinnit väreille ja fonteille. Ainoa lisävalintamahdollisuus sivustolla oli poistaa minimoinnin yhteydessä myös ne osittain vanhentuneet selainkohtaiset määritykset, joita ei enää pitäisi tarvita CSS3:a tukevissa selaimissa. Tätä testatessani muutamien sivujen marginaalit menivät hieman poskelleen. Asian selvittämiseksi kannattaa tehdä matalimman tason minimointi sekä oletusasetuksin että poistaen CSS3:ssa vanhentuneet määritykset. Tällöin sisältö on edelleen useammalla kuin yhdellä rivillä, joten sen vertailu riveittäin onnistuu helposti esimerkiksi Compare It! -ohjelmalla [7]. Kyseinen ohjelma vertaa kahta tiedostoa riveittäin ja näyttää, millä riveillä tiedostot eroavat toisistaan.

Vertailun avulla selviää, että olen määrittänyt kahdessa eri kohdassa marginaaleja niin, että olen käyttänyt ensin lyhennysmerkintää, jossa kahdella arvolla määrätään neljä marginaalia ja sen jälkeen määrännyt uudestaan vielä yhden marginaalin. Minimointi poisti tämän jälkimmäisen määrityksen kokonaan. Tyyli piti siis muokata niin, että käytetäänkin kolmea arvoa neljälle marginaalille, koska vasen ja oikea olivat juuri ne samanarvoiset. Sen jälkeen CSS3:ssa vanhentuneiden määritysten poisto toimi ongelmitta. Samaisen vertailun avulla nähdään myös ne tyylimääritykset, joille tiedostossa oli vielä selainkohtaisetkin määritykset vanhojen versioiden tukemiseksi. Näitä arvoja olivat box-shadow, box-sizing, opacity sekä text-size-adjust.

4.1.3 CSS3-standardin tyylien selaintuet

CSS3:n mukaisten ominaisuuksien selaintuet nähdään W3Schools-sivuston ylläpitämästä listauksesta [10], joka kertoo seuraavat taulukon 5 mukaiset selaintuet niille ominaisuuksille, joita ForeAmmatti käyttää. Taulukosta selviää selaimen versio, josta alkaen se on tukenut universaalia tyylimääritystä, ja joissain tapauksissa myös se versio, joka on tukenut kyseistä ominaisuutta jo aiemmin, mutta vaatinut selainkohtaisen etuliitteen.

Taulukko 5. *ForeAmmatin käyttämien CSS3-määritteiden selaintuki.*

	Chrome	**Explorer**	**Firefox**	**Safari**	**Opera**
box-shadow	10.0 (4.0 -webkit-)	9.0	4.0 (3.5 -moz-)	5.1 (3.1 -webkit-)	10.5
box-sizing	10.0 (4.0 -webkit-)	8.0	29.0 (2.0 -moz-)	5.1 (3.2 -webkit-)	9.5
opacity	4.0	9.0	2.0	3.1	9.0

ForeAmmatti ei tue kovin vanhoja selaimia, vaan pyrimme olemaan mukana tuoreessa kehityksessä. Tämän vuoden alussa lopetimme eniten ongelmia tuottaneiden vanhojen Presto-pohjaisten Opera-selainten sekä Internet Explorerin versiota 9 vanhempien versioiden tukemisen. Opera 15.0 julkaistiin kokonaan uudelleen kirjoitettuna 2.7.2013 ja vanhan Presto-moottorin sijaan selain kirjoitettiin WebKit-moottorin päälle. Tämän riittävän uuden Operan saa asennettua esimerkiksi Windows XP, OS X 10.6 ja Linux-käyttöjärjestelmiin [23]. Opera ei siis aiheuta ongelmia CSS3-määritysten tuen suhteen.

Internet Explorer 9 julkaistiin jo 14.3.2011 ja sen sai asennettua Windows Vistaan ja Windows Server 2008:aan, joihin oli asennettu Service Pack 2 sekä suoraan Windows 7:aan ja Windows Server 2008 R2:een [26][27]. Mahdolliset Windows XP ja Windows Server 2003:n käyttäjät joutuvat siis käyttämään jotakin muuta selainta. Internet Explorerin uudemmat versiot eivät myöskään aiheuta ongelmia CSS3-määritysten tuen suhteen.

Muiden valmistajien selaimien osalta julkaisupäivät ovat seuraavat: Google Chrome 8.3.2011 (versio 10.0.648) [19], Firefox 29.4.2014 (versio 29.0) [22] ja Safari 20.7.2011 (versio 5.1) [34]. Hieman yllättäen myöhäisin päivämäärä on Firefox-selaimen 29.4.2014. Toki etuliitteellä Firefoxikin on tukenut box-sizing-ominaisuutta jo versiosta 2.0 ja onhan version 29 julkaisustakin nyt aikaa jo yli 18 kuukautta. Toistaiseksi jätetään myös etuliitteein varustetut määritykset CSS-tyylitiedostoon ainakin muutaman kuukauden ajaksi.

Text-size-adjust ei ole ollenkaan standardin mukainen määrite eikä sillä ole vaikutusta työpöytäselaimiin, ellei sen arvoksi määritetä ”none”, joka estää WebKit-pohjaisissa työpöytäselaimissa, kuten Chrome ja Safari, sivun koon muuttamisen. Määritteen tarkoituksena on sen sijaan ohjata mobiiliselaimia näyttämään teksti aina 100 %:n koossa, eli siinä koossa, jossa sivun tekijä on sen tarkoittanut näkyvän. Tätä määritettä pitää aina käyttää selainkohtaisten etuliitteiden kanssa. [38]

4.2 JavaScript-kooditiedostot

Kuten jo valitsimia optimoitaessa huomattiin, että voimassa olevasta JavaScript-koodista suurin osa oli jäänyt HTML-koodin joukkoon, vaikka tarvetta sille ei olisikaan ollut. Lisäksi yhdessä ulkoisessa JavaScript-tiedostossa on runsaasti sellaisia funktioita, joita ei enää hiljattain tehdyn ForeAmmatin täydellisen ulkoasun uudistamisen myötä tarvita. Tarkoitus on poistaa ylimääräiset funktiot, siirtää kaikki mahdollinen koodi ulkoisiin tiedostoihin sekä jakaa koodi kolmeen ulkoiseen tiedostoon niin, että yhdessä on valitsimiin liittyvä osuus, toisessa blogiin liittyvä osuus ja kolmannessa sivuston yhteinen osuus.

ForeAmmatin yleiseen JavaScript-tiedostoon jää siis tällä hetkellä ulkoasuun, responsiivisuuteen ja lomakkeiden täyttöön ja muotoiluun liittyviä funktioita. Lisäksi sieltä löytyy myös muutamia automaattisesti suoritettavia tarkistusfunktioita, joilla korjataan tietyissä laite- ja selainyhdistelmissä esiintyviä renderöintivirheitä.

Funktiot myös kirjoitetaan kokonaan uudelleen käyttämään apunaan jQuery-kirjastoa perinteisen JavaScriptin sijaan, koska moni asia onnistuu pienemmällä merkkimäärällä jQueryn avulla. Eroa voi joskus olla jopa kymmeniä merkkejä. Esimerkiksi sopii tilanne, jossa pitää tehdä jotakin kaikille niille elementeille, joilla on sama tyyliluokka.

```
var rivit = document.getElementsByClassName('taso1_rivi');
for (var i=0; i<rivit.length; i++) {
	rivit[i].setAttribute('style', 'color: white; background-color: rgb(79,129,189)');
}
```

Kuva 11. *Valkoisen fontin ja ”ForeSinisen” taustavärin asettaminen ennen.*

Jos haluat asettaa perinteisellä JavaScriptillä esimerkiksi valkoisen fonttivärin ja ”ForeSinisen” taustavärin niille elementeille, joiden tyyliluokka on taso1_rivi, pitää nämä elementit poimia kuvan 11 rivillä yksi olevalla koodilla. Sen jälkeen tulee tehdä silmukkarakenne, joka kiertää koko äsken tallennetun vektorin läpi ja asettaa jokaisen elementin style-attribuutin arvoksi halutut tyylimääritykset.

```
$('.taso1_rivi').css({'color': 'white', 'backgroundColor': 'rgb(79,129,189)' });
```

Kuva 12. *Valkoisen fontin ja ”ForeSinisen” taustavärin asettaminen jälkeen.*

Kun sama koodi kirjoitetaan käyttäen hyväksi jQuery-kirjastoa, niin kaikki elementit saadaan poimittua tavallisella luokkavalitsimella, eli samalla, jolla CSS-tiedostossakin tyylit asetetaan. Jos muutos kohdistuu kaikkiin vektorin elementteihin, ei erillistä silmukkaa tarvitse kirjoittaa, vaan se on oletustoiminto, ja voidaan suoraan asettaa halutut tyylimääritykset css-metodilla.

4.2.1 JavaScript-koodin jakaminen osiin

Blogisivujen JavaScript-tiedostoon tallennetaan valitsimista tutut ammattien ID-arvot sekä viralliset nimet ja blogia kirjoittavien henkilöiden esittelytekstit. Nämä toistuvat joka kirjoituksessa, kun kirjoittajan esittely tulee kirjoituksen loppuun ja sivuun linkkejä aiheeseen liittyvistä ammateista. Aiemmin aiheeseen liittyvät ammatit linkitettiin niin, että piti itse kirjoittaa linkki, ammatin ID sekä nimi. Nyt riittää, että kutsutaan vain funktiota, joko kirjoittajan nimellä tai ammatin ID-arvolla, jolloin funktio kirjoittaa lopputuloksena kyseisen tekstin tai linkin. Blogisivujen JavaScript-tiedosto sisältää vain maksuttoman ForeAmmatin 200 ammattia, koska ainoastaan niihin viitataan blogikirjoituksissa.

Valitsimien JavaScript-tiedostoon tallennetaan edellisten ammattitietojen lisäksi myös tarvittavia totuusarvoja sekä ammattiluokituksen toinen taso ja kaikki loput ammatit sekä ELY-alueet ID-arvoineen. Lisäksi tähän tiedostoon tulee monia funktioita, joita tarvitaan valitsimien esittämiseen, piilottamiseen ja korostamiseen eri väreillä. Jokaisella sivulla ladataan siis yhdestä kahteen JavaScript-tiedostoa, koska yksittäisellä sivulla esiintyy

vain joko blogiin liittyvä kirjoitus tai ammatti- ja aluevalitsimet, mutta ei koskaan molempia yhtä aikaa.

4.2.2 JSF.js -kirjasto

AJAX-mekanismien käyttäminen aiheuttaa JSF:n yhteydessä sen, että ladattavien tiedostojen joukkoon lisätään automaattisesti jsf.js-niminen aputiedosto javax.faces-kirjastosta [16]. Tiesin jo aiemminkin sen, minkä kyseinen Safarin ohje kertoo, eli tämän kirjaston voi myös ladata eksplisiittisesti erillisellä komennolla. Mutta mikäli tämä erillinen komento ei suoraan vastaa tietynlaista script-elementtiä, tai käytetty koodi ei sitä tuota, tekniikka lisää sen automaattisesti toisen kerran. Tämän työn yhteydessä aioin muuttaa myös kaikkien JavaScript-tiedostojen lataamisen asynkroniseksi, jolloin esimerkiksi erittäin hitailla yhteyksillä, ruudulle saataisiin nopeammin näkymään jotakin tekstiä, ja kirjastojen latauduttua suoritettaisiin vasta ulkoasun muotoiluja ynnä muita. Mutta en löytänyt keinoa ladata jsf.js-tiedostoa asynkronisesti, vaan sen script-elementin ilmestyminen toiseen kertaan pakotti aina latauksen synkronisena.

Tämän takaiskun jälkeen luovuin myös muiden sivuston kannalta välttämättömien JavaScript-tiedostojen asynkronisen lataamisen tavoittelusta. Sitä varten olisi kuitenkin tarvittu runsaasti lisää koodia ja ehtoja, koska tiedostojen lataukset olisivat voineet päättyä milloin tahansa ja missä tahansa järjestyksessä. Myöskään dokumentin jäsentämisen tila ei olisi riippunut niistä, eli kaikki tilanteet olisi pitänyt ottaa huomioon. Todennäköisesti dokumentti olisi ollut valmis ensin ja sen jälkeen tiedostot kokojärjestyksessä, mutta ei siitä takeita olisi ollut. Viimeistään tiedostojen välimuistista lataaminen olisi sekoittanut kaikki. Jokaisessa funktiokutsussa olisi siis pitänyt tarkistaa, että onko kohde jo olemassa, onko dokumentin jäsentäminen jo tarpeeksi pitkällä ja niin edelleen, riippuen siitä, mitä ja missä tilanteessa olisi tarvittu.

Lopulta olin varsin tyytyväinen nykyiseenkin ratkaisuun, jossa ulkoasun kannalta tärkeät tiedostot, kuten jQuery-kirjasto, ladataan synkronisesti, jolloin odotusaika on hitailla yhteyksillä hieman pidempi, mutta käyttäjä saa heti valmiin näköisen sivun eteensä. Itse en ainakaan henkilökohtaisesti pidä sivuista, joiden ulkoasu muuttuu jopa useampaan kertaan sitä ladattaessa ja siksi tarjoankin mielelläni myös meidän sivustomme käyttäjille suoraan valmiilta näyttävän sivun.

4.2.3 JavaScript-koodien minimointi

JavaScript-koodin minimoinnilla tarkoitetaan sellaista koneellista operaatiota, jolla se muutetaan selkeälukuisesta muodosta tiiviimpään, mutta silti edelleen toimivaan muotoon [43]. Tyypillisimpiä itse sisältöön vaikuttavia muutoksia ovat esimerkiksi muuttujien esittelyt, joita voidaan tehdä aina useita kerralla, jos tietotyypit ovat samoja [28]. JavaScript ei kuitenkaan ole tyypitetty kieli eli muuttujilla ei ole erityisiä tietotyyppejä,

kuten kokonaisluku, merkkijono ja niin edelleen, vaan muuttujat ovat aina muuttujia. Samoin yksinkertaiset toimenpiteet, kuten vain yhden muuttujan arvon muuttaminen ehtolausekkeessa, voidaan toteuttaa ilman erillistä aaltosulkein kirjoitettua lohkoa.

Etsin Googlen hakukoneen avulla muun muassa termillä ”crunch” tai ”cruncher”, joka on yleisesti käytetty termi, kun tarkoitetaan minimointia tai tiivistämistä. Kokeilin sattumanvaraisesti yhdeksää erilaista löytämääni sivustoa, joista kuusi antoi toimivalta näyttäviä tuloksia. Alkujaan ForeAmmatin JavaScript-koodin tiedoston koko oli 53 419 tavua ja tulokset vaihtelivat välillä 33 697 - 42 306 tavua. Ensimmäisenä ajattelin kokeilla tietysti pienimmän koon antanutta JavaScript Compressor -sivustoa [28], joka osoittautuikin toimivaksi minimoinniksi. Sivuston JavaScript-koodien toiminta pysyi muuttumattomana. Tämä onkin ulkoasultaan hyvin samanlainen sivusto kuin aiemmin CSS:n minimointiin käytetty ja nämä ovatkin ilmeisesti saman tahon tekemiä, koska molemmissa on linkki toiseen.

Oletusasetuksilla tiivistäminen on tehokkainta ja turvallisinta. Sivusto ei kerro tarkasti, mitä kaikkea valitut optimoinnit tekevät, mutta ainakin kaikki kommentit ja saavuttamattomat funktiot poistetaan sekä muuttujien esittelyt ja funktioiden järjestykset optimoidaan. Ainoat lisävalintamahdollisuudet sivustolla olivat käyttää ei-turvallisia muunnoksia sekä vielä enemmän ei-turvallisia optimointeja. Ensin mainittu ei vaikuta minimointituloksen tavumäärään lainkaan ja toinenkin vähentää vain kaksi tavua, joten en lähtenyt näiden vaikutusta toimintaan testailemaan koettuani ne tarpeettomiksi.

4.3 Kuvatiedostot

Etusivun kuudesta kuvatiedostosta viisi ladataan meidän palvelimeltamme. Kuvat ovat suurimmaksi osaksi ikonityyppisiä ja hyvin pieniä. Yhteensä näillä viidellä kuvalla on kokoa vain 9 086 tavua. Jokaista kuvaa joudutaan siis pyytämään erillisellä GET-paketilla. Kun tutkitaan näiden pyyntöjen otsikkotiedoista aiheutuvaa tiedonsiirtoa, huomataan, että niiden koko on keskimäärin noin 490 tavua per tiedosto. Vastauspaketeissa on luonnollisesti myös otsikkotiedot, ja niiden koko on keskimäärin noin 335 tavua per tiedosto. Yhteensä näistä tulee siis keskimäärin noin 825 tavua per tiedosto. Esimerkiksi pienimmän kuvan koko on vain 138 tavua, joten hyötykuormaa on vain noin 14,3 %.

Tarkoituksena on ensin kokeilla miten paljon kuvien tiedostokoot pienenevät, kun ne pakataan teknisesti ottaen häviöllisesti, mutta niin, ettei ihmissilmä huomaa eroa. Sen jälkeen pienimmät kuvat saadaan toivottavasti liitettyä osaksi CSS-tiedostoa, jotta ladattavien tiedostojen määrää voidaan pienentää. Samalla toki CSS-tiedoston koko kasvaa, mutta muutos on kannattava, mikäli kokonaishyöty pois jäävistä otsikkotiedoista jää suuremmaksi kuin CSS-tiedoston kasvu, joka joudutaan joka tapauksessa lataamaan.

4.3.1 Kuvien pakkaaminen

Jokin aika sitten sain erään postituslistan kautta mainoksen TinyPNG-sivustosta [39] ja kysyi, että onko tällaisesta meille hyötyä. Lupasin tutkia asiaa. TinyPNG kertoo käyttävänsä kvantisointia suorittaessaan png- tai jpeg-tiedostojen häviöllistä pakkausta. Työkalu vähentää valikoidusti kuvan värien määrää, jolloin sen tallentamiseen riittää tietysti pienempi määrä tavuja. Värien määrän vähentämisen myötä esimerkiksi 24-bittinen png-tiedosto voidaan konvertoida 8-bittiseksi indeksoiduksi värikuvaksi. Muutoksen luvataan olevan tiedostokoossa hyvin suuri, mutta ihmissilmälle olematon [39]. Tein muutamia kokeiluja ja väite piti paikkansa. Tiedostokoko pienentyi, enkä havainnut kuvissa mitään eroa alkuperäiseen, vaikka suurensin esimerkiksi 24*24 kokoisen symbolin näytön kokoiseksi.

Google tarjoaa PageSpeed-sivustonsa ja -liitännäisensä [32] myötä myös muun muassa kuvien optimointia. Tässä tosin tuntuu olevan jokin prosenttiraja. Mikäli muutosprosentti ei ylitä raja-arvoa, PageSpeed ei ehdota sille kuvalle optimointia ollenkaan. Päätin testata sekä PageSpeedin että TinyPNG:n avulla yhdeksää sivustollamme esiintyvää pienehköä kuvaa, joiden suurinkin alkuperäiskoko jää alle 20 kilotavun.

Taulukko 6. *Kuvien koot alun perin, PageSpeedin ja TinyPNG:n jälkeen.*

	Alkuperäinen [tavua]	**PageSpeed [tavua]**	**TinyPNG [tavua]**
Facebook-ikoni	311		272
PLUS-vinjetti	5 202	3 264	1 541
PLUS-logo	1 025		455
KTI selite	2 834	1 089	1 024
KTI akseli	621		621
PRO puhekupla	19 890	12 646	5 626
Twitter-ikoni	421		396
Luokitteluperustemenu	138		100
Luokitteluperustevalinta	3 014	295	207

Kuten taulukosta 6 huomataan, niin TinyPNG osoittautui hyväksi työkaluksi. PageSpeed-liitännäinen ehdotti optimointia vain niille neljälle kuvalle, joiden koot löytyvät taulukosta. Se ei kuitenkaan pystynyt mihinkään sellaiseen, johon ei TinyPNG olisi myös pystynyt. Jos tallensi ensin PageSpeedin optimoiman kuvan ja syötti sen TinyPNG:lle, oli lopputulos vastaava, kuin suoraan alkuperäisellä tiedostolla. Jos taas käytti ensin TinyPNG:tä, ei PageSpeed ehdottanut enää mitään optimointeja.

4.3.2 Kuvien Base64-koodaaminen

Kuvien pieneen hyötykuorman osuuteen lähdin kokeilemaan Base64-koodausta, jonka avulla binääritiedostot (kuten kuvat) voidaan koodata ASCII-merkeiksi. Merkeiksi on va-

littu sekä suuret että pienet kirjaimet väliltä A - Z (26 + 26 kpl), kaikki kymmenen numeroa sekä plusmerkki ja kauttaviiva. Nämä 64 merkkiä voidaan esittää kuuden bitin avulla (2^6 = 64 erilaista binääriarvoa). Normaalisti esimerkiksi teksti koodataan ASCII-muodossa, jolloin yksi merkki vie yhden tavun eli kahdeksan bittiä. Kolmen kirjaimen ASCII-muotoinen koodaus vie siis 24 bittiä. Tämä jaetaan kuuden bitin jaksoihin, jolloin desimaalista indeksiä vastaava luku on välillä 0 - 63. Indeksin mukainen merkki sijoitetaan kuuden bitin välein Base64-koodattuun merkkijonoon. Tästä johtuen kolmesta merkistä tulee tietysti neljä, koska bittimäärä laskee kahdeksasta kuuteen. Base64-koodattu merkkijono on siis pituudeltaan neljä kolmasosaa ASCII-koodatusta merkkijonosta.

Sama pätee myös binääritiedostoihin, eli kuvan esittäminen Base64-koodattuna merkkijonona vie binääritasolla enemmän tilaa. Kuitenkin tämä merkkijono voidaan liittää CSS-tyylitiedoston osaksi ja välttää näin erillisen kuvatiedoston siirrosta aiheutuvat otsikkotiedot. Tyypillisessä tapauksessa tämä muunnettu kuva asetetaan esimerkiksi div-elementin taustakuvaksi ja sille määritellään korkeus- ja leveys alkuperäisen kuvatiedoston mukaan. Nyt voidaan muodostaa laskukaava, jolla voidaan selvittää tyypillinen raja-arvo, jota pienempi kuvatiedosto kannattaa koodata Base64-koodauksella.

$$FS + 825 = \frac{4}{3}FS + 65$$

Kuva 13. *Base64-koodauksen kannattavuus verrattuna kuvatiedostoon.*

Base64-koodauksen kannattavuus selviää kuvan 13 mukaisella laskukaavalla, jossa FS tarkoittaa kuvatiedoston kokoa. Base64:n myötä kolmesta kuvatiedoston tavusta syntyy siis neljä tavua Base64-koodia. Kuvatiedostoon liittyy keskimäärin 825 tavua otsikkotietoja ja Base64-koodiin keskimäärin 65 tavua CSS-tyyliä. Yhtälöstä ratkaistaan FS, ja saadaan lopputulokseksi FS = 2280. Tämä tarkoittaa sitä, että jos kuvatiedoston koko kasvaa 2 280 tavua suuremmaksi, Base64-koodaus ei välttämättä ole enää kannattavaa.

Taulukko 7. *Kuvien koot kuvatiedostoina sekä Base64-koodattuina.*

	Kuvana [tavua]	**Base64:nä [tavua]**	**Säästö**
Facebook-ikoni	1 097	440	59,9 %
PLUS-vinjetti	2 366	2 105	11,0 %
PLUS-logo	1 280	668	47,8 %
KTI selite	1 849	1 446	21,8 %
KTI akseli	1 446	877	39,3 %
PRO puhekupla	6 451	7 583	-17,5 %
Twitter-ikoni	1 221	604	50,5 %
Luokitteluperustemenu	925	212	77,1 %
Luokitteluperustevalinta	1 032	352	65,9 %

Taulukkoon 7 on koottu näiden yhdeksän kuvan koko kuvatiedostona siirrettäessä TinyPNG-työkalun jälkeen, eli edellisen taulukon TinyPNG-kokoon on lisätty otsikkotietojen keskiarvo 825 tavua per tiedosto. Base64-kokoon on puolestaan lisätty se CSS-koodi, jonka kuvan esittäminen alkuperäisen näköisenä vaatii. Kuvien muuttaminen Base64-koodiksi tehtiin Base64 Image Encoder -sivuston avulla [6]. Osassa tilanteista riittää pelkkä taustakuvan määritteleminen, mutta osassa pitää antaa myös div-elementille ne mitat, jotka kuvatiedostollakin ovat. Vinjetin tapauksessa ei tarvita ollenkaan enempää CSS-koodia kuin kuvatiedostonkaan kanssa, ja siksi sen koodaaminen antaa vielä säästöä, vaikka koko olikin hieman laskettua raja-arvoa suurempi. Kaavan perusteella on nyt helppo arvata, että yli 3 000 tavuisen kuvatiedoston Base64-koodaaminen lienee enää hyvin harvoin kannattavaa. Tällöin tulisi olla tilanne, jossa ylimääräistä CSS:ää ei tarvita ja otsikkotiedot yltäisivät 1 000 tavun kokoon.

Taulukon 7 perusteella voidaan siis vetää se johtopäätös, että tarpeeksi pienien kuvien esittäminen Base64-koodauksen avulla on järkevämpää kuin kuvatiedostona lataaminen. Lisäksi tällä säästetään yhteyksien määrää, jonka enimmäismäärä on rajoitettu. Tästä kuitenkin lisää myöhemmin alaluvussa 4.4.1. Tässä tilanteessa puhekupla tullaan pitämään kuvatiedostona, ja muut koodataan CSS:n joukkoon Base64-koodauksella. Näin ollen yhteensä tiedonsiirrossa säästetään kuvien osalta 4 512 tavua sekä kahdeksan yhteyttä, kun ainoastaan puhekupla ladataan kuvatiedostona ja muut Base64-koodattuna.

4.3.3 Blogikuvien pakkaaminen

Kaikki aiemmin käsitellyt kuvatiedostot olivat png-kuvia, johon TinyPNG-sivuston työkalu oli alun perin suunniteltukin. Blogissamme olevat kuvat ovat sen sijaan jpeg-kuvia, mutta TinyPNG väittää osaavansa myös niiden pakkaamisen. ForeAmmatin blogissa on tätä kirjoittaessani 15 kirjoitusta, jotka sisältävät yhteensä 21 erilaista kuvaa. Jokaiseen kirjoitukseen liittyy Facebookin suositusten mukainen 1200*630 kokoinen kuva, sen lisäksi löytyy kolme liitekuvaa sekä kirjoittajien 80*80 kokoiset esittelykuvat. En kuitenkaan käy kaikkia kuvia läpi erikseen vaan esitän pelkät keskiarvot ryhmittäin.

Taulukko 8. *Blogikuvien keskimääräiset koot alun perin ja pakattuina.*

	Alun perin [tavua]	TinyPNG [tavua]	Säästö
Kirjoitusten kuvat (15 kpl)	150 002	118 647	20,9 %
Liitekuvat (3 kpl)	67 561	35 034	48,1 %
Kirjoittajien kuvat (3 kpl)	23 250	2 874	87,6 %

Kuten taulukosta 8 voidaan todeta, että TinyPNG-työkalu osaa pakata hyvin myös jpeg-tiedostoja. Suurten pääkuvien keskimääräinen koko pienentyi noin 20,9 %, liitekuvien noin 48,1 % sekä kirjoittajien kuvien noin 87,6 %. Tiedostojen välillä on tietysti värien määrästä riippuen suurta vaihtelua ja pakkausprosentteja löytyy laidasta laitaan, sekä alle 10 % että yli 90 %. Yhteensä kaikkien 21 kuvan koko pieneni 629 037 tavua, joka tekee

noin 24,9 %. Voidaan siis todeta, että jatkossa uuden kirjoituksen yhteydessä myös uusi kuva kannattaa kierrättää TinyPNG-työkalun kautta.

4.4 Liitännäiset

ForeAmmatti käyttää useita liitännäisiä eri lähteistä. Liitännäisellä tarkoitan tässä yhteydessä muuta kuin itse kirjoittamaani koodia. Suurin osa näiden koodista liittyy joko kokonaan tai suurelta osin JavaScriptiin. Monissa näistä apukirjastoista on valmis koodi, jolla liitännäisen pystyy liittämään sitä käyttävälle sivustolle tilanteessa kuin tilanteessa. Näin mekin olemme aiemmin toimineet, mutta nyt on tarkoitus tutustua siihen, miten nämä apukirjastot ladataan ja liitetään juuri meidän sivustomme kannalta tehokkaasti.

Samassa yhteydessä toteutetaan tietysti samanlaisia muutoksia kuin aiemminkin omien koodien yhteydessä, eli koodi kirjoitetaan lyhyemmäksi, jos se vain on mahdollista. Nämäkin koodit minimoidaan ja liitetään tapauskohtaisesti vain ne liitännäiset, jotka juuri sen hetkisellä sivulla tarvitaan.

4.4.1 jQuery-kirjasto

jQuery-kirjasto on yksi oleellisimmista liitännäisistä sivustollamme. Sen avulla muun muassa hienosäädämme sivuston ulkonäköä niin, että se näyttää hyvältä kaikilla päätelaitteilla. Lisäksi käytämme myös joitakin sen tarjoamia tehokeinoja, kuten esiin tai piiloon liukuvat elementit. jQuery-kirjastoa käyttää myös Highcharts-grafiikkakirjasto. Olemme aiemminkin käyttäneet jo jQueryn tuotantokäyttöön tarkoitettua valmiiksi minimoitua versiota. Minimoidun version 2.1.4 koko on 84 345 tavua ja sen lataaminen meidän palvelimeltamme kestää noin 100 - 110 ms selaimella mitattuna sekä 90 - 120 ms ApacheBenchillä mitattuna. Tämän resurssin lataus siirretään nyt meidän palvelimeltamme jQueryn CDN:lle (Content Delivery Network), josta tämä resurssi toivottavasti latautuu vielä nopeammin. jQuery CDN:stä toimitettu versio on 34 418 tavun kokoiseksi gzipillä pakattu versio, ja sen lataaminen ilman salattua yhteyttä kestää noin 50 - 70 ms selaimella mitattuna sekä 140 - 160 ms ApacheBenchillä mitattuna. Aikaero johtuu siitä, ettei ApacheBench käytä oletusasetuksilla pakkausta hyväkseen [12]. Siitäkään ei ole tietoa, priorisoiko jQuery CDN tällaisen robotin huonommin kuin ihmiskäyttäjän. Mitataan asiaa uudestaan myöhemmin, kun myös ForeAmmatti käyttää GZIP-pakkausta.

Resurssin lataamisesta eri verkko-osoitteesta olisi kuitenkin muutakin hyötyä kuin mahdollinen nopeushyöty. Yleisimmät nykyiset selaimet sallivat samanaikaisesti kuusi yhteyttä samaan verkko-osoitteeseen. Nopealla yhteydellä dokumentti on yleensä ehtinyt jo latautua kokonaan ennen kuin sen jäsentäminen on edennyt niin pitkälle, että resurssien lataaminen alkaisi, mutta hitaammalla yhteydellä myös resurssit latautuvat samaan aikaan dokumentin kanssa. Tämä tarkoittaa sitä, että vain viittä resurssia voidaan ruveta lataamaan samaan aikaan, mikäli dokumentin lataus on vielä kesken. Jos resursseja on enemmän ja dokumentin jäsentämisen puolesta niiden lataus voisi jo alkaa, resurssit voivat

joutua odottamaan muiden resurssien loppuun latautumista. Tilanne korostuu esimerkiksi erittäin hitailla mobiiliyhteyksillä, jolloin hiemankaan suurempien resurssien latausajat voivat venyä pitkiksi. Tässä on yksi syy lisää, jonka takia pienet kuvat oli kannattavaa Base64-koodata, jolloin säästetään otsikkotietojen ohella myös yhteysmääriä. jQuery on meidän tapauksessamme ainoita resursseja, jotka voitaisiin ladata myös oman palvelimemme ulkopuolelta, joten siksi tätä asiaa kannattaa tutkia. Tähän palataan myöhemmin.

4.4.2 Highcharts-grafiikkakirjastot

Highcharts-grafiikkakirjastojen optimoinnissa päädyttiin koko kirjastoperheen kustomointiin juuri meidän tarpeisiimme sopivaksi. Kolmesta tiedostosta saatiin muodostettua vain yksi tiedosto, jonka koko jää pienemmäksi kuin kolmen tiedoston summa. Samalla tämän tiedoston lataaminen ja käyttöönotto piti pohtia täysin uudelleen. Tämä kohtalaisen suuri muutos ansaitsee siten täysin oman lukunsa, joten Highchartsin optimointia käsittelenkin tarkemmin seuraavassa luvussa.

4.4.3 Google Maps API

ForeAmmatti esittää muun muassa työpaikkojen tarkan sijainnin kartalla Google Maps JavaScript API:n avulla. Tämä liitännäinen toimii niin, että ensin sivustolle liitetään viittaus JavaScript-koodiin, joka asettaa muutamia muuttujia ennen kuin koodi kirjoittaa varsinaisen main.js-tiedoston script-elementin sivustolle document.write-metodilla. Kun main.js-tiedosto on latautunut, voidaan vasta aloittaa kartan piirtäminen, jonka jälkeen main.js-tiedosto sitten pyytää lataamaan useita muita tiedostoja, joita kartan piirtäminen vaatii. Tiedostolla on kuitenkin kokoa pakattunakin 19 380 tavua ja alustuskoodilla 978 tavua, joten sivun hahmontamista hidastaa yli 20 kilotavun kuorma.

Koska Googlen tekemä koodi kirjoittaa main.js-tiedoston script-elementin suoraan dokumenttiin, sitä ei voi ladata asynkronisesti. Koska document.write-metodi kirjoittaa sisältönsä heti kutsunsa jälkeen, niin dokumentin jäsentäminen ei voi jatkua ilman kyseisen tiedoston lataamista. Sen sijaan koodin tulisi luoda uusi script-elementti, asettaa sen parametrit ja sen jälkeen asettaa se esimerkiksi head-elementin lapsielementiksi, jolloin dokumentin jäsentäminen voisi jatkua samaan aikaan eteenpäin. Tämä saattaa kuitenkin olla tarkoituksellisesti estetty Googlen toimesta, ja mahdollinen syy selviää hetken kuluttua.

Google Mapsin dokumentaatio neuvoo, miten edellä mainitut tiedostot ladataan asynkronisesti [3]. Dokumentaatio esittää, että sivulle lisätään onload-tapahtumaan kuuntelija, ja aloitetaan Google Maps -tiedostojen lataus vasta sen jälkeen, kun dokumentti on jo kokonaan valmis. Teknisesti ottaen tämä ei kuitenkaan ole asynkronista lataamista vaan pikemminkin vain latausajankohdan siirtämistä myöhäisemmäksi. Hitaalla yhteydellä tästä aiheutuu se, että dokumentti voi olla jo valmistunut, mutta kartalle varattu elementti pysyy pitkään tyhjänä. Tämä ratkaisu ei kuulostanut hyvältä, joten ajattelin kokeilla jotain muuta.

Kopioin Google Mapsin alustuskoodit ForeAmmatin JavaScript-tiedostoihin ja kirjoitin document.write-metodin tilalle edellä kuvatulla tavalla uuden script-elementin luonnin, jolle pystyin nyt määräämään myös asynkronisen latauksen ja kohteeksi tietysti samainen main.js-tiedosto. Mikä siis oli se mahdollinen synkroniseen lataukseen pakottamisen syy? Vaikuttaa siltä, ettei itse kartan piirtämisen käynnistävää funktiota saa kutsua ennen kuin koko dokumentin lataus on valmis. Vaikka DOMContentLoaded-tapahtuma [11] olisi jo lauennut, kartan määrittelytiedot siis jäsennetty ja main.js-tiedosto ladattu, niin silti aina välillä virhekonsoliin tulostuu virheilmoitus, joka viittaa siihen, että jotakin elementtiä yritettäisiin käyttää ennen kuin se on olemassa. Koska main.js:n koodi on kuitenkin minimoitu ja funktioiden nimet ja parametrit hämärretty, niin koodista on todella hankalaa selvittää, mitä se yrittää tehdä. Virheilmoitus oli siinä mielessä outo, koska kaikkien elementtien pitäisi tässä vaiheessa olla olemassa. Lisäksi virheilmoitus tulee useimmin silloin, jos main.js ladataankin selaimen välimuistista eikä suoraan palvelimelta. Vaikuttaa siis siltä, että jos main.js on valmiina liian paljon ennen dokumenttia, niin tapahtuu jokin virhe. Lopulta ratkaisin asian niin, että main.js:n latauksen valmistuttua tarkistetaan dokumentin tila. Jos se on jo kokonaan valmis, voidaan kutsua kartan piirtävää funktiota. Jos ei, kiinnitetään onload-tapahtumaan kuuntelija, joka käynnistää kartan piirtämisen sen jälkeen, kun dokumentti on kokonaan valmis. Näin toimien virheilmoituksia ei tule, mutta tietysti joissakin tilanteissa kartan piirtäminen odottaa nytkin turhaan, mutta ainakin tuo reilu 20 kilotavua saadaan ladattua dokumentin jäsentämisen kanssa rinnakkain.

Muutaman viikon kuluttua näiden muutosten tekemisen jälkeen selvisi se todellinen syy, miksi Google Maps API pitää ladata synkronisesti. Google tekee myös alustuskoodiin muutoksia, joka muuttaa siihen sisällytettyjä versionumeroita ja samalla estää vanhojen versioiden toiminnan. Alustuskoodeja ei siis voi kopioida, vaan ne on liitettävä tiedostosta, jonka sisältämä document.write-metodi estää asynkronisen latauksen. Ainoa vaihtoehto sivun jäsentämisen estämisen poistoon on siis edellä mainitun Googlen dokumentaation mukainen ”asynkroninen” lataaminen eli se, että Google Maps API:n tiedostojen lataus aloitetaan vasta onload-tapahtuman lauettua.

4.4.4 Muut liitännäiset

Muita liitännäisiä ovat muun muassa Google Analytics, Inspectlet, Twitter sekä kaksi erilaista Facebook-liitännäistä. Nämä latautuvatkin jo oletuksena asynkronisesti, mutta kaikissa on hyvin samanlainen koodi, joka luo script-elementin ja liittää sen kohdeosoitteeksi liitännäisen osoitteen. Koodeissa on myös toiminnan kannalta ylimääräistä, kuten script-elementtien ID-arvot, joten ne voidaan poistaa ja koodia voidaan hieman yksinkertaistaa niin, että se on erillisessä funktiossa, jota voidaan jokaisen liitännäisen kohdalla kutsua vain kohdeosoite parametrina. Tällä säästetään myös hieman merkkejä ja lisäksi saadaan liitettyä kätevästi joka sivulle vain yksi rivi per tarvittava liitännäinen.

5. HIGHCHARTS-GRAFIIKKAKIRJASTOJEN KÄYTÖN OPTIMOINTI

Highcharts on interaktiivinen grafiikkakirjasto, jota ForeAmmatti käyttää tiedon visualisoimiseksi. Highcharts ladataan JavaScript-tiedostoista ja erilaisten kuvaajien luominen onnistuu melko helposti. Yksinkertaisessa tapauksessa tarvitaan noin 10 - 20 erilaista määritystä sekä piirrettäväksi tarkoitettu datasisältö. Koodi muodostaa määritysten pohjalta SVG-elementin sitä tukevissa uusissa nykyaikaisissa nettiselaimissa ja piirtää vektorigrafiikan avulla kaiken kuvaajan tarvitseman sisällön, joka on muun muassa täysin leveyden mukaan skaalautuvaa. Käyttäjän kanssa tapahtuva vuorovaikutus tapahtuu JavaScriptin avulla. [21]

5.1 Tarkempi esittely

Olemme käyttäneet peruskirjastoa jo pitkään tavanomaisten pylväs- ja viivakaavioiden piirtämiseen. Highcharts-kirjastoperhe on kuitenkin sittemmin kasvanut ja laajentunut huomattavasti ja peruskirjaston lisäksi ladattavissa on nykyään neljä erilaista perusosaa. Toinen on itse Highchartsin lisäkirjasto, josta käytämme muun muassa nopeusmittarin tyylistä määrää osoittavaa pyöreää kuvaajaa. Kolmas on Highstock-niminen pääasiassa aikasarjojen esittämiseen käytettävä kirjasto, jota emme ole vielä tarvinneet. Neljäs on Highmaps-niminen karttojen esittämiseen käytettävä kirjasto. Karttoja emme ole piirtäneet tällä kirjastolla, mutta tämä sisältää myös heatmap-tyylisen kuvaajan, jollainen lisättiin ForeAmmattiin viime kesänä. Heatmap-kirjaston tosin pystyy lataamaan myös erillisenä moduulina.

Highcharts-kirjastojen kustomointimahdollisuus [20] on lisätty heidän sivuilleen hiljattain. Sen avulla pystyy valitsemaan ne ominaisuudet, joita itse tarvitsee, ja muodostamaan koodin valintojensa mukaan. Kirjastoihin sisältyy monia sellaisiakin kuvaajia ja muita ominaisuuksia, joita emme tarvitse. Toisaalta, osa ominaisuuksista vaatii myös muiden ominaisuuksien läsnäoloa. Tarkoitus olisi nyt tutkia kirjaston kustomointimahdollisuutta ja erilaisten ominaisuuksien vaikutusta tiedostokokoon. Samalla kolmen erillisen tiedoston lataamisesta päästäisiin vain yhteen tiedostoon. Tuloksien perusteella voidaan sitten myöhemmin arvioida, voisiko jotkin kuvaajat toteuttaa eri tavalla, jos sillä säästäisi huomattavan määrän tiedostokokoa.

5.2 Alkutilanne

Tiedostokokoja vertaillaan kahdessa muodossa, jotka ovat kehittäjämuoto ja tuotantomuoto. Kehittäjämuodossa JavaScript-koodi on jäsenneltyä ja helppolukuista sekä selkeästi kommentoitua. Tällöin mahdollisten virheiden löytäminen ja tutkiminen on helppoa,

mutta tämä muoto tarvitsee paljon enemmän tilaa. Siksi se onkin tarkoitettu käytettäväksi vain esimerkiksi yrityksen omassa testiympäristössä, jossa tiedostokoot voivat olla suurempia. Tuotantomuoto on tarkoitettu käytettäväksi sovelluksissa, jotka ovat loppukäyttäjien käytettävissä. Se on pitkälle minimoitu versio samasta JavaScript-koodista, josta on poistettu kommentit ja ylimääräiset välilyönnit sekä rivinvaihdot. Rivien pituudet ovat tässä tapauksessa keskimäärin 500 merkkiä. Kirjoitushetkellä uusimmat versiot ovat 12. kesäkuuta 2015 julkaistut versiot Highcharts 4.1.6, Highstock 2.1.6 ja Highmaps 1.1.6.

Taulukko 9. *ForeAmmatin käyttämät Highcharts-kirjastot alkutilanteessa.*

Kirjasto	**Kehittäjämuodon koko**	**Tuotantomuodon koko**
Highcharts-peruskirjasto	500 637 tavua	162 232 tavua
Highcharts-lisäkirjasto	66 570 tavua	24 002 tavua
Heatmap-lisäosa	16 913 tavua	7 875 tavua
Yhteensä	**584 120 tavua**	**194 109 tavua**

Alkutilanteessa ForeAmmatissa käytetään siis kolmen erillisen tiedoston tuotantomuotoa, joiden avulla käytetyt kuvaajat piirretään. Yhteensä nämä tiedostot ovat kooltaan taulukon 9 mukaisesti 194 109 tavua.

5.3 Kustomointi

Kustomointi aloitetaan määrittämällä perustaso, joka riippuu siitä, halutaanko Highchartsia ajaa jQueryn päällä vai käytetäänkö sen omaa viitekehystä. Kehittäjämuodossakin oma viitekehys kasvattaa tiedostokokoa vain noin kolme kilotavua, mutta koska ForeAmmatti käyttää jQuery-kirjastoa muuhunkin, sen lataaminen on joka tapauksessa pakollista. Tällöin voidaan myös Highchartsia ajaa jQueryn päällä. Perustaso sisältää Highchartsin ytimen, joka tarvitaan kaikkeen muuhun sekä viivakuvaajan, jota ei pysty valinnoista poistamaan.

Seuraavissa taulukoissa tutkitaan ForeAmmatin vaatimien ominaisuuksien aiheuttamaa tiedostokoon lisäystä. Ensimmäinen sarake kertoo ominaisuuden, jotka esitellään tarkemmin taulukoiden jälkeen. Toinen sarake kertoo kehittäjämuodon tiedostokoon tavuina, kolmas lisäyksen prosentteina verrattuna perustasoon ja neljäs tuotantomuodon tiedostokoon tavuina sekä viides lisäyksen prosentteina verrattuna perustasoon.

Taulukko 10. *Highchartsin kustomointi ForeAmmatin tarpeisiin, osa 1.*

Ominaisuus	**Km [B]**	**Km [%]**	**Tm [B]**	**Tm [%]**
Perustaso	301 830	0,00	97 063	0,00
Html	310 062	2,73	99 798	2,82
Plotlines or bands	307 599	1,91	99 212	2,21
Stacking	310 473	2,86	99 955	2,98
Datalabels	323 109	7,05	103 255	6,38

Taulukko 11. *Highchartsin kustomointi ForeAmmatin tarpeisiin, osa 2.*

Interaction	342 178	13,37	112 177	15,57
Tooltip (+ interaction)	356 227	18,02	116 868	20,40
Column	312 419	3,51	100 389	3,43
Bar (+ column)	312 663	3,59	100 462	3,50
Pie	314 861	4,32	101 661	4,74
Columnrange (+ area + column + arearange)	326 218	8,08	105 259	8,44
Gauge (+ plotlines or bands)	329 974	9,32	107 754	11,01
Heatmap (+ column + scatter)	330 203	9,40	108 835	12,13
Yhteensä	**476 807**	**57,97**	**160 113**	**64,96**

Vaikka suurin osa ominaisuuksien nimistä taulukoissa 10 ja 11 onkin yksikäsitteisiä, niin käydään ne kuitenkin lyhyesti läpi. Ensimmäiset neljä ominaisuutta liittyvät kuvaajien yleiseen käyttäytymiseen. Html-ominaisuuden avulla voidaan renderöidä työkaluvihjeen sisältö HTML-koodina SVG-elementtien sijaan. Tämä sallii monimutkaisemmat muotoilut työkaluvihjeeseen kuten taulukot ja kuvat. Plotlines or bands -ominaisuudella voidaan muun muassa värittää kaavioalueen taustasta osa eri värillä. Stacking-ominaisuudella sarjat voidaan pinota toistensa päälle sen sijaan että ne esitettäisiin vierekkäin. Datalabels-ominaisuuden avulla voidaan esittää datapisteiden arvoja, esim. pylväs- tai viivakaavion tapauksessa datapisteen tarkka arvo tai ympyräkaaviossa sektorin prosentuaalinen koko.

Seuraavat kaksi ominaisuutta liittyvät dynaamisuuteen ja vuorovaikutukseen. Interaction-ominaisuuden avulla voidaan aktivoida kaavion ja hiiren välinen vuorovaikutus, joka saa aikaan esimerkiksi hiirellä osoitetun pylvään korostumisen paksummalla reunaviivalla. Tooltip-ominaisuus ottaa käyttöön työkaluvihjeen, kun kohdetta osoitetaan hiirellä. Tätä ei luonnollisestikaan voi ottaa käyttöön ilman interaction-ominaisuutta.

Seuraavat viisi ominaisuutta ovat kaaviotyyppejä. Column on perinteinen pylväskaavio. Bar on palkkikaavio eli käytännössä pylväskaavio, jonka x- ja y-akselit on käännetty toisinpäin. Siksi palkkikaavio myös vaatii pylväskaavion. Pie tarkoittaa yleisesti prosenttien esittämiseen käytettyä ympyräkaaviota. Columnrange on mielestäni hieman hassu nimitys kaaviolle, joka on käytännössä kyljelleen käännetty pylväskaavio eli palkkikaavio, jossa on käytössä pinoaminen, mutta kaikki arvot eivät ala samasta pisteestä. ForeAmmatti käyttää tätä kaaviota esittämään ammatin palkkajakaumaa. Koska ammatin minimipalkka ei ole nolla tai kaikissa ammateissa sama, niin pinottu palkkikaavio ei sovellu tämän tiedon esittämiseen. Columnrange-kaavio kuitenkin vaatii vastaavasti toimivan aluekaavion, nimeltään arearange sekä luonnollisesti perinteiset pylväs- ja aluekaaviot. Aiemmin mainittu nopeusmittaria esittävä kaavio on nimeltään gauge ja se vaatii myös plotlines or bands -ominaisuuden.

Viimeinen ominaisuus löytyy moduulien listalta. Heatmap on uusin kaaviotyyppi, joka on otettu käyttöön ForeAmmatissa. Sen käyttäminen vaatii myös pylväs- ja pistekaaviot.

Heatmap-kaaviolla voidaan esittää dataa ikään kuin xy-koordinaatistossa, jolloin jokaiselle arvopisteelle tulee esimerkiksi samankokoinen ruutu, jonka väri tai värin tummuus voi jo luoda ensivaikutelman itse arvon suuruudesta.

Kuten taulukoista 10 ja 11 huomataan, että interaction- ja tooltip-ominaisuudet ovat selvästi tiedostokooltaan kookkaimmat. Tämä selittynee sillä, että Highcharts sisältänee paljon JavaScript-koodia erilaisille kaaviotyypeille, joissa korostetaan hiiren osoittamaa sarjaa tai avataan kohteen työkaluvihje. Yksittäisten kaavioiden kuten columnrange, gauge ja heatmap korkea prosentti selittyy yksinkertaisesti sillä, että ne vaativat useita muita ominaisuuksia toimiakseen.

5.4 Kustomoinnin lopputulos

Sen sijaan että luopuisimme työkaluvihjeistä, ajatuksena on juuri päinvastainen. Aiomme tulevaisuudessa lisätä työkaluvihjeiden käyttöä ja määrää entisestään ja monipuolistaa niiden sisältöä, jotta niistä olisi mahdollisimman paljon hyötyä loppukäyttäjille. Tässä vaiheessa emme yritä jättää mitään ominaisuuksia pois, vaan pidämme toistaiseksi kaikki tarvittavat ominaisuudet kustomoidussa kirjastossa.

Lopuksi voidaan lukea taulukon 11 viimeiseltä riviltä, että nykyisen ForeAmmatin vaatimilla ominaisuuksilla kustomoidun Highcharts-kirjaston tuotantomuodon tiedostokoko on 160 113 tavua. Kun sitä verrataan alkutilanteen kolmen eri tiedoston summasta muodostuvaan vastaavaan kokoon, joka oli 194 109 tavua, todetaan säästöä syntyneen 33 996 tavua. Highcharts-tiedostojen koko pienentyi siis noin 17,51 %. Samalla tietysti myös tiedonsiirrossa säästetään lisäksi kahden tiedoston otsikkotiedot.

5.5 Lataamisen optimointi

Tiedosto on siis pakkaamattomana noin 160 kilotavun kokoinen, joten sen lataaminen kannattaa aloittaa mahdollisimman pian. Latauksen päätyttyä pitää kuitenkin varmistua siitä, että myös jQuery on ehtinyt latautumaan, jotta sen päälle rakentuva Highcharts voidaan jäsentää. Kuvaajat voidaan piirtää heti, kun Highcharts-kirjasto on jäsennetty toisin kuin Google Mapsin tapauksessa. Saattaa kuitenkin olla niin, että dokumentista on vielä lataamatta se osa, joka mahdollisesti muokkaa ulkoasua, mikäli ikkunan leveys on sellaisella välillä, että ulkoasu muuttuu vielä lataamisen jälkeen.

Mikäli käyttäjällä sattuu tämän levyinen selainikkuna olemaan, niin kuvaajat eivät osaa automaattisesti leventyä tai kaventua mahdollisesti muuttuneisiin isäntäelementteihinsä niiden piirtämisen jälkeen. Tästä syystä lisätään vielä tarkistus dokumentin valmistumisesta, eli jos dokumentti ei ollut vielä täysin valmis, kiinnitetään onload-tapahtumaan kuuntelija, joka kutsuu kuvaajien piirtämisfunktiota uudelleen sinä hetkenä. Tällöin kuvaajien leveydet mukautuvat varmasti oikeisiin elementtikokoihin.

6. TIEDOSTOJEN JAKAMISEN OPTIMOINTI

Ulkoisten tiedostojen sisältöjen optimoimisen jälkeen ne kaikki sisältävät vain tarpeellista informaatiota ja ne on minimoitu mahdollisimman tiiviiseen muotoon ja pieneen tiedostokokoon. Tarkoituksena on tutkia vielä XHTML-tiedostojen minimointia sekä kahta muuta merkittävää tiedostojen jakamisen optimointiin liittyvää asiaa eli siirrettävän kuorman pakkaamista sekä selaimien välimuistien toimintaa.

Selaimet tukevat siirrettävän kuorman pakkaamista GZIP:n avulla täysin automaattisesti ilman, että käyttäjä huomaa mitään. Pakkaaminen ja purkaminen vaativat kuitenkin suoritinaikaa sekä lähettäjältä että vastaanottajalta, mutta toisaalta siirrettävän datan määrä on silloin pienempi. Selaimien välimuistien hyödyntämisellä voidaan estää niiden tietojen lataamista yhä uudelleen ja uudelleen, jotka ovat pysyneet edellisenkin sivun latauksen jälkeen samoina. Selain tulee kuitenkin saada lataamaan uudet tiedot heti, jos ne ovat muuttuneet.

6.1 XHTML-tiedostojen minimointi

XHTML-tiedostojen minimoinnilla tarkoitetaan sellaista koneellista operaatiota, jolla se muutetaan selkeälukuisesta muodosta tiiviimpään, mutta silti edelleen toimivaan muotoon. XHTML-tiedostojen erittäin tarkka syntaksi ei salli minkäänlaista väärää merkkiä mihinkään kohtaan. HTML-tiedostoista poiketen vaaditaan täsmällisesti oikeanlaiset lainausmerkit pareittain, aloitus- ja lopetustagit pareittain oikeassa järjestyksessä, attribuuttien arvot eivät saa olla tyhjiä ja niin edelleen. XHTML-tiedosto muuntuu palvelimen käsittelyn aikana HTML-merkkaukseksi XML (eXtensible Markup Language) -jäsentimen avulla. Käyttäjälle palautettava vastaus on siis aina HTML-muodossa. Oletusarvoisesti XML-jäsennin jäsentää koko XHTML-tiedoston sisällön ja muodostaa siitä puumaisen rakenteen. Mikäli XHTML-tiedoston sisäinen JavaScript-koodi sisältää <- tai &-merkin, siitä aiheutuu XML-jäsennysvirhe. <-merkki tarkoittaa uuden elementin alkua ja &-merkki merkkientiteettiä eli numeerisen koodin avulla syötettyä merkkiä. Näitä merkkejä sisältävä JavaScript-koodi voidaan merkitä niin sanotuksi CDATA-osuudeksi (ei-jäsennettäväksi merkkidataksi), jolloin XML-jäsennin ohittaa sen, ja JavaScript-koodi toimii eikä aiheuta virheitä [47].

Etsin Googlen hakukoneen avulla muun muassa termillä ”crunch” tai ”cruncher”, joka on yleisesti käytetty termi, kun tarkoitetaan minimointia tai tiivistämistä. Kokeilin sattumanvaraisesti yhdeksää erilaista löytämääni sivustoa, joista viisi antoi ylipäätään mitään tuloksia. ForeAmmatin osaamisiin keskittyvän sivun XHTML-tiedoston koko oli alun perin 43 174 tavua ja tulokset vaihtelivat välillä 24 833 - 25 729 tavua. Kaksi pienimmän tu-

loksen antanutta ei osannut lukea XHTML:n elementtejä oikein, ja ne tekivät sivusta toimimattoman. Ensimmäinen toimiva minimointi oli kolmannen sivuston antama, josta lopputuloksen kooksi muodostui 25 365 tavua. Vaikka sivuston minimoija onkin tarkoitettu JavaScript-koodille, niin se ei ainakaan muuttanut XHTML:n elementtejä niin, ettei-vät ne enää toimisi. Kyseinen sivusto on Webmasters Cavernin JavaScript Cruncher [29]. Mikään testatuista sivustoista ei kuitenkaan osaa XHTML:n syntaksia siinä määrin, että ne ymmärtäisivät CDATA-osuuksien merkityksen. Tiedostot olisi tietysti voinut minimoida useassa osassa, mutta se ei tuntunut kovin mielekkäältä työltä. Löysin kuitenkin ongelmaani Stack Overflow -yhteisön sivuilta Michael Ridleyn kirjoittaman ja Robin Winslowin muokkaaman selityksen [44], joka kertoi, että CDATA-osuus tarvitaan vain, jos XML-jäsentimen läpi menevässä dokumentissa halutaan JavaScript-koodissa kirjoittaa <- ja &-merkkejä niiden entiteettivastineiden sijaan. Käänsin siis asian päinvastoin, ja muutin pienempi kuin -merkit entiteetiksi ”< ” sekä suurempi kuin -merkit entiteetiksi ”> ” ja &-merkit entiteetiksi ”& ”. Nyt JavaScript-koodi toimii ilman CDATA-osuudeksi merkitsemistä, jolloin myös edellä mainittu minimointi toimii koko tiedoston sisältöön kerralla.

6.2 GZIP-pakkaus

Java-sovelluspalvelimemme Tomcat tukee GZIP-pakkauksen käyttöönottoa muutamalla parametrilla, jotka lisätään palvelimen tietoihin kohtaan, joka kuuntelee HTTP-liikenteen oletusporttia 80. Pakkauksen käyttöönotto tapahtuu MIME-tietotyypeittäin ja sen oletusarvona on puhdas teksti, HTML-sivut sekä CSS-tiedostot. Yleensä binääritiedostot kuten kuvat, musiikit ja videot ovat jo pakattuja, eikä niiden pakkaaminen uudelleen ole kannattavaa [4]. Kolmen hallitsevan tietotyypin eli HTML:n, CSS:n ja JavaScriptin kolmikko on hyvin semanttista eli se sisältää paljon toistoa, joten sen pakkaaminen on oletettavasti kannattavinta.

Pakotetaan ensin GZIP-pakkaus käyttöön kaikille tietotyypeille ja kaiken kokoisille tiedostoille, jotta voidaan arvioida pakkauksen tarpeellisuutta.

Taulukko 12. *GZIP-pakkauksen vaikutus tiedostotyypeittäin.*

Kuvaus	MIME	Pakkausmäärä	Pakkausprosentti
HTML-sivut	text/html	27,4 - 54,5 kt	73,2 - 76,2 %
CSS-tiedosto	text/css	20,5 kt	65,7 %
JavaScript-tiedostot	application/javascript	5,5 - 100 kt	63,4 - 74,3 %
Blogikirjoitusten kuvat	image/jpeg	204 - 2 814 t	0,15 - 2,77 %
Blogin kirjoittajien kuvat	image/jpeg	42 - 48 t	1,37 - 1,80 %
PRO puhekupla	image/png	-18 t	-0,32 %

Kuten taulukosta 12 voidaan todeta, että oletetun kolmikon pakkaaminen näyttää tarpeelliselta ja järkevältä. Kuvat oli valmiiksi minimoitu alaluvussa 4.3, eikä niiden pakkaaminen uudelleen ole tarpeellista, koska parhaimmassakin tapauksessa niiden pakkausprosentti jää 2,77 prosenttiin. Vajaan 110 kilotavun kuvan koko pienenisi alle kolme kilotavua. Tiedostokoon pienentyminen tuntuu niin alhaiselta, etten näe järkevänä tutkia kuvien pakkaamisen vaikutusta suoritinkäyttöön. Otetaan pakkaus käyttöön vain kolmelle semanttista sisältöä sisältävälle tietotyypille, joiden pakkausprosentit vaihtelevat välillä 63,4 - 76,2 %.

GZIP-pakkauksen käyttöönoton jälkeen testataan jQuery-kirjaston lataamista uudelleen. Todetaan kuitenkin melko pian, ettei edes kaikkien niiden otsikkotietojen lisääminen ApacheBenchiin, jotka nettiselain lähettää, saa jQuery CDN -palvelinta palauttamaan tiedostoa GZIP-pakattuna. Meidän palvelimella riittää ohjeen [12] mukainen yksi otsikkokenttä, joka kertoo palvelimelle, että käyttäjä ymmärtää GZIP-pakattua dataa. Toinen mielenkiintoinen yksityiskohta on tiedostokokojen vertaileminen nettiselaimen kautta. ApacheBenchillä pystyttiin toteamaan, että kummaltakin palvelimelta pakkaamattomana ladatun tiedoston koko on luonnollisestikin sama 84 345 tavua. GZIP-pakattuna tiedoston koko on kuitenkin jQuery CDN -palvelimelta ladattuna 34 418 tavua ja meidän palvelimeltamme 29 569 tavua. Meidän palvelimeltamme ladattuna pakattu tiedosto on siis noin 14 % pienempi. Tomcatin dokumentaatiot eivät kerro, että se tukisi pakkaustason säätämistä GZIP-pakkauksen yhteydessä, mutta ilmeisesti jQuery CDN:n käyttämä sovelluspalvelin tukee, joten ero johtunee pakkaustasoista.

Taulukko 13. *jQuery-kirjaston lataaminen ilman pakkausta.*

Palvelin	**jQuery CDN**	**ForeAmmatti**
Yhteyden muodostaminen	41,27 ms ± 1,44 ms	16,36 ms ± 1,13 ms
Odottaminen / Prosessointi	39,91 ms ± 8,57 ms	16,00 ms ± 1,06 ms
Tiedonsiirto	50,01 ms ± 12,10 ms	58,48 ms ± 3,53 ms
Yhteensä	**131,19 ms**	**90,84 ms**

Palvelimien eroa mitataan ApacheBenchillä niin, että kummaltakin palvelimelta ladataan pakkaamaton jQuery-kirjasto sata kertaa yksi kerrallaan. Tulokset löytyvät taulukosta 13. Yhteyden muodostamisajan keskiarvo jQuery CDN -palvelimeen on 41,27 ms ja keskihajonta 1,44 ms. Tiedonsiirron keskiarvo on 50,01 ms ja keskihajonta 12,10 ms. Meidän palvelimeltamme ladattuna yhteyden muodostamisajan keskiarvo on 16,36 ms ja keskihajonta 1,13 ms. Tiedonsiirron keskiarvo on 58,48 ms ja keskihajonta 3,53 ms. Yhteensä nämä tekevät siis 91,28 ms ja 74,84 ms. Eroa syntyy siis meidän palvelimen hyväksi 16,44 ms eli noin 18 %. Lisäksi palvelimen suorittama muu prosessointi kestää jQuery CDN -palvelimelta keskimäärin 39,91 ms ja meidän palvelimelta 16,00 ms. Kaikkien kestot yhteensä ovat siis 131,19 ms ja 90,84 ms. Eroa siis jälleen meidän palvelimen hyväksi 40,35 ms eli noin 30,8 %.

Kun mitataan pakattujen tiedostojen latausaikaa nettiselaimella, jQuery CDN -palvelimelta tiedosto latautuu noin 45 - 55 millisekunnissa ja meidän palvelimeltamme noin 85 - 95 millisekunnissa. jQuery CDN -palvelin vaihtaa kahden eri IP-osoitteen välillä viiden minuutin välein. DNS-kysely näyttää kuluttavan yleensä noin 22 ms, joten tämä selittää pidemmän yhteyden muodostamisajan ApacheBench-testissä. Selaimella suoritetussa testissä jQuery CDN -palvelin on reilusti nopeampi, joten heidän käyttämänsä pakkaustapa on varmaankin nopeampi, ja pakkaustaso matalampi, koska sen lopputuloksena tiedostokokokin on hieman isompi. Yksi vaihtoehto on myös, että jQuery CDN -palvelimella tiedostot ovat valmiiksi pakattuja. Tomcat ei kuitenkaan tue muuta kuin lennossa tapahtuvaa pakkaamista. DNS-kyselynkin kanssa jQuery CDN -palvelin on hieman nopeampi, joten päädytään lataamaan jQuery-kirjasto jatkossa sieltä.

6.3 Selaimien välimuistit

Vaikka tiedostojen koot pienenivätkin sekä tämän työn että pakkauksen käyttöönoton myötä huomattavasti, ladattavaa riittää silti vielä paljon. Oletuksena Tomcat-palvelin lisää vastauspakettien otsikkotietoihin sekä Last-Modified että ETag -kentät, joista ensin mainitulla se viestittää, milloin tiedostoa on viimeksi muokattu ja jälkimmäisellä, mikä on tiedoston sisällöstä laskettu avain. Selaimen tulisi näin ollen liittää pyyntöpakettien otsikkotietoihin sekä If-Modified-Since että If-None-Match -kentät, jos kyseinen tiedosto löytyy jo sen välimuistista. Mikäli tiedostoa ei ole muokattu kyseisen ajan jälkeen ja sisällön avain vastaa myös kyseistä oikeaa arvoa, palvelimen tulisi lähettää vain lyhyt vastaus 304-pakettina (Not Modified), jolloin selain saa luvan käyttää välimuistissaan olevaa tiedostoa. [5]

Näiden molempien toiminta on osoittautunut hieman epävarmaksi, erityisesti niin, että selain ei oikeista vastauksista huolimatta hae muuttunutta tiedostoa palvelimelta, vaikka sen pitäisi. Tämän työn aikana sain myös useasti ohjaustiedot sekoamaan niin, että pakotettuani muuttuneen resurssin hakemisen palvelimelta, niin selain haki sitä myöhemminkin joka kerta, vaikka se ei olisi muuttunut. Toinen syy välimuistien käytön parantamiselle on jatkuvasti lisääntyvä mobiilikäyttö. En juurikaan näe järkeä siinä, että samoihin resursseihin liittyen kysytään joka sivulla, onko tiedosto ehtinyt muuttua. Erityisesti hitailla ja runsaasti viivettä sisältävillä mobiiliverkoilla monien yhteyksien avaaminen tätä kyselyä varten on täysin turhaa.

Tomcat sisältää ExpiresFilter-nimisen Servletin, johon on koodattu Apache-palvelimen lisäosan mod_expires innoittamana vastaavat toiminnallisuudet. ExpiresFilter lisää sekä Expires että Max-Age kentät vastauspakettien otsikkotietoihin. Määrityksiin voi antaa joko suoran ajan tai käyttää laskennallista viittausta joko pyyntöhetken tai tiedoston muokkauspäivän suhteen [15]. Meidän käyttöömme sopii malli, jossa tiedostojen nimen perään lisätään aikaleimaa vastaava numerosarja, jolloin resurssin voimassaoloaika välimuistissa saa olla hyvin pitkä. Tällöin resurssi ladataan tyypillisesti välimuistiin vain ensimmäisen pyynnön yhteydessä ja samaa tiedostoa voidaan välimuistista käyttää niin

kauan kunnes se korvataan uudella. Aikaleiman arvoa muutetaan, kun tiedostoon tulee muutoksia, jolloin selain pyytää joka tapauksessa aina uutta tiedostoa. Tämä metodi tuntuu myös toimivan oikein joka tilanteessa, ja verkkoliikennettäkin säästyy, kun tämä tapa ei kommunikoi ollenkaan palvelimen kanssa, mikäli tiedosto löytyy välimuistista eikä sen voimassaoloaika ole ylittynyt [5].

Aikaleima voidaan siis lisätä kuviin, tyylitiedostoihin sekä JavaScript-koodien tiedostoihin. Näiden voimassaoloajaksi voidaan määrittää esimerkiksi kymmenen vuotta muokkauspäivämäärästä laskien. Tiedostot, jotka muuttuvat muutaman kerran kuukaudessa ovat haastavampia. Etukäteen ei voida tietää tapahtuuko muutos käyttäjän ensimmäisen pyynnön jälkeen tunnin vai viikon kuluttua. Asetetaan siis näille pdf-dokumenteille ja Excel-tiedostoille sama määritys kuin dynaamisille HTML-sivuillekin eli nolla sekuntia pyyntöhetkestä lukien. Tällöin vanhentumisaika on siis sama kuin pyynnön aika eli välimuistia ei käytetä näille tiedostoille sekä sivuille. Lisäksi otsikkopalkin pieni 16*16 kuva (favicon.ico) tulee olla aina samanniminen, joten lisätään sen ehdoksi vaikkapa kymmenen tuntia pyyntöhetkestä lukien. Nyt jos tätä kuvaa vaihdetaan, niin muutos näkyy käyttäjällä viimeistään kymmenen tunnin kuluttua, mutta samalla estetään kuitenkin tiedoston lataaminen uudelleen joka sivulla. Vaikka itse kuvan koko onkin vain 509 tavua, mutta ne aiemminkin mainitut otsikkotiedot ovat tässäkin tapauksessa suuressa roolissa.

6.4 Tuleva HTTP/2-protokolla

Sovelluspalvelimien tuki uudelle HTTP/2-protokollalle [24] on juuri yleistymässä tätä työtä kirjoitettaessa. HTTP/2 perustuu standardoimattomaan SPDY-protokollaan [36], jonka Google esitteli jo vuonna 2009. Tuleva HTTP/2-protokolla pyrkii ratkaisemaan ongelmia, joita tässäkin työssä ratkottiin. Alun perin 1990-luvun alussa julkaistu HTTP/1.0 sai kaipaamansa päivityksen vuonna 1999, jolloin HTTP/1.1 julkaistiin. Sen jälkeen Internet-sivustot ovat kuitenkin muuttuneet rajusti, kun sivustoista on tullut entistä monipuolisempia ja interaktiivisempia. Resurssitiedostojen määrä on kasvanut kymmeniin, jopa satoihin tiedostoihin samalla, kun nopeat nettiyhteydet ovat tulleet kaikkien saataville. [25]

Eräs suurimmista HTTP/1.1:n haittapuolista on se, että yhdellä yhteydellä voi ladata vain yhtä tiedostoa kerrallaan [25]. Selainvalmistajat ovat auttaneet asiaa hieman sallimalla tyypillisesti kuudesta kahdeksaan yhteyttä verkko-osoitetta kohden. Tästä kerroin jo aiemmin alaluvussa 6.2, kun siirsin jQuery-kirjaston latauksen pois omalta palvelimeltamme säästääkseni yhden palvelimellemme tulevan yhteyden. Vastaavanlaisia kiertoteitä moni sivustonkehittäjä pohtii koko ajan kiertääkseen HTTP/1.1:n rajoituksia. HTTP/2 tulee tarjoamaan oletettavasti melko tehokkaita keinoja näihin ongelmakohtiin.

HTTP/2-protokollaa käytettäessä käyttäjän ja palvelimen välille muodostetaan vain yksi yhteys, ja kaikki tiedostot siirretään samalla yhteydellä. HTTP/2 tarjoaa myös otsikkotietojen pakkaamisen sekä binäärikoodauksen nykyisten selkokielisten ja pakkaamattomien

otsikkotietojen sijaan. Varsinkin, kun nykyään nämä otsikkotiedot siirretään jokaisen tiedoston kohdalla erikseen, odotettavissa on hyötyä senkin suhteen. Koska salattujen yhteyksien julkisten avaimien laskennat ovat hyvin suoritinintensiivisiä operaatioita, niin salattujen yhteyksien käyttö ja niiden muodostamiseen vaadittavat kättelyt ovat raskaita toimenpiteitä. [25]

Vaikka HTTP/2:n määrittely ei varsinaisesti vaadi salauksen käyttämistä, selainvalmistajat ovat kuitenkin toteuttaneet HTTP/2-protokollan tuen ainoastaan salatun protokollan yli. Tämän asian ei odoteta muuttuvan ainakaan lähitulevaisuudessa [25]. Löytämäni HTTP/2-protokollan suorituskykyvertailun [35] ja kaiken muun lukemani perusteella olen vakuuttunut HTTP/2:n hyödyistä. Varsinkin meillä Suomessa tyypillisillä asymmetrisillä nettiyhteyksillä GET-pakettien otsikkotietojen pakkaaminen, vain yhden yhteyden käyttö ja kaikkien tiedostojen lataaminen yhdellä yhteydellä riittävät luullakseni muodostamaan riittävän suuren hyödyn, jotta HTTP/2-protokollaan kannattaa siirtyä.

Meidän tapauksessamme olemme muutenkin jo suunnitelleet salauksen käyttöönottamista tulevaisuudessa, joka taas lisäisi sivuston odotusaikoja ainakin HTTP/1.1-versiota käytettäessä. Tarkoituksenani on siis suorittaa lähikuukausina testauksia NGINX-välityspalvelimen avulla HTTP/2-protokollaa hyödyntäen. Lisäksi HTTP/2-tukea ollaan parhaillaan kehittämässä myös Tomcat-sovelluspalvelimeen ja se toivottavasti nähdään jo seuraavan suuremman päivityksen jälkeen versiossa yhdeksän. HTTP/2-protokollan myötä palataan taas osittain vanhaankin, kun resurssitiedostoja ei enää kannata jakaa niin moneen osaan, koska se hankaloittaa päivittämistä. Myöskään jQuery-kirjastoa ei kannattaisi silloin enää ladata ulkopuolelta, koska jokainen verkko-osoite vaatii oman yhteytensä. Tällöin mahdollisimman monta tiedostoa kannattaisi ladata omalta palvelimelta yhteyksien säästämiseksi.

7. LOPPUTULOS

Kaikkien muutosten jälkeen ajetaan uudelleen samat mittaukset kuin alkutilanteessakin. Nyt kaikista tuloksista on kaksi eri versiota, joista ensimmäinen vastaa kenen tahansa ihmiskäyttäjän näkemiä tuloksia. Toisessa samat mittaukset ajetaan niin, että Java-koodissa on tunnistettu HTTP-pyynnöstä User Agent -merkkijono, joka kertoo millaisesta selaimesta tai laitteesta on kyse. Tämän perusteella rajoitetaan tietojen hakemista ja näkymistä, jos kyseessä on jokin web-crawler tai muu robotiksi rinnastettava laite. Näihin luetaan myös testeissä käyttämäni ApacheBench sekä Load Impact -kuormitustesti.

7.1 Etusivun resurssit

Samoja selainten kehittäjätyökaluja ja niiden lisäosia käyttäen, mitataan tiedostojen lataamiseen kuluvat ajat sekä niiden tarkat koot. Suurin osa tarvittavista resursseista ladataan alkutilanteen hetkellä yksinkertaisemman koodin takia. Sisäisesti HTML-koodi muodostuu useista XHTML-tiedostoista, joilla saadaan aikaan se, että yhdelle sivulle liitetään vain tarvittavat lisäosat. Etusivulta jäi nyt pois esimerkiksi Highcharts-grafiikkakirjasto sekä Google Maps API:n tiedostot. Päädyin lopulta tähän ratkaisuun sen takia, että suurin osa kuitenkin aloittaa etusivulta, joten pyritään näyttämään se nopeasti. Muille sivuille siirryttäessä yhteiset resurssit ovat jo välimuistissa, jolloin voidaan ladata tarpeelliset muut kirjastot. Mikäli käyttäjä luo istunnon esimerkiksi linkin avulla suoraan sellaiselle sivulle, jossa esimerkiksi grafiikkakirjastoa tarvitaan, tällöin latausaika on tietysti pidempi, mutta se on oletettavissakin. Näissä mittauksissa on kytketty selaimen välimuisti pois käytöstä, jotta kaikki tiedostot ladataan uudelleen joka kerta, jotta saadaan luotettavia keskiarvoaikoja. Latausaikojen kestot mitataan taaskin kymmenen kerran keskiarvona.

Taulukko 14. *Koot ja ajat yhteensä ihmiskäyttäjälle lopuksi.*

Sisällön tyyppi	**Koko yhteensä**	**Aika yhteensä**
XHTML-sivu (1 kpl)	9 047 tavua	108,7 ms
Pyydetty CSS (2 kpl)	11 009 tavua	79,0 ms
Fontit edellisiin (2 kpl)	35 088 tavua	36,3 ms
Pyydetty JS (7 kpl)	134 232 tavua	320,1 ms
Sisältö edellisiin (2 kpl)	394 tavua	13,5 ms
Kuvat (6 kpl)	563 tavua	20,8 ms
Yhteensä (20 kpl)	**190 333 tavua**	**578,4 ms**

Sivun sisältö on jaettu edelleen kuuteen eri ryhmään taulukon 14 mukaisesti. Ensimmäisenä on itse XHTML-sivu, jonka koko on pienentynyt arvosta 420 998 tavua arvoon 9 047 tavua (-411 951 tavua / -97,9 %). Latausaika on pienentynyt arvosta 431,7 ms ar-

voon 108,7 ms (-323,0 ms / -74,8 %). Pyydetyt tyylitiedostot (CSS) ovat edelleen ForeAmmatin tyylitiedosto sekä Roboto Slab -fontti, jonka pyyntö ohjautuu Googlen palvelimelle. Näiden koko on pienentynyt arvosta 30 162 tavua arvoon 11 009 tavua (-19 153 tavua / -63,5 %) ja latausaika on kasvanut arvosta 75,3 ms arvoon 79,0 ms (+3,7 ms / +4,9 %). Fontit ovat samat fontit kahdella eri paksuudella kuin alussa. Näidenkin koko on pienentynyt arvosta 43 400 tavua arvoon 35 088 tavua (-8 312 tavua / -19,2 %) ja latausaika arvosta 80,3 ms arvoon 36,3 ms (-44,0 ms / -54,8 %), vaikka näihinhän ei tehty mitään muutoksia. Oletan, että Google tekee myös aktiivisesti koko ajan vastaavaa työtä kuin minäkin ja yrittää säästää tiedonsiirtomäärissä kaiken mahdollisen.

Pyydettyihin JavaScript-komentosarjoihin (JS) jää muun muassa itse tehtyä JavaScript-koodia kaksi tiedostoa, jQuery-kirjasto sekä JSF:n kirjasto. Google Analytics -lisäosa sekä Facebook-lisäosa ladataan asynkronisesti ja Inspectlet-lisäosan lataus alkaa vasta onload-tapahtuman jälkeen. Näiden koko on pienentynyt arvosta 332 210 tavua arvoon 134 232 tavua (-197 978 tavua / -59,6 %). Latausaika on pienentynyt arvosta 762,8 ms arvoon 320,1 ms (-442,7 ms / -58,0 %). Seuraavaan ryhmään jää vain edellisen ryhmän JavaScript-lisäosien autentikointeja. Näiden koko on pienentynyt arvosta 112 006 tavua arvoon 394 tavua (-111 612 tavua / -99,6 %). Latausaika on pienentynyt arvosta 207,0 ms arvoon 13,5 ms (-193,5 ms / -93,5 %). Ladattaviksi kuviksi jää nyt vain favicon.ico (osoiterivillä ja suosikeissa näkyvä pieni ikoni), koska muut kuvat ovat Base64-koodattu CSS-tiedoston sisään. Ladattavien kuvien tiedostokoko on pienentynyt arvosta 9 129 tavua arvoon 563 tavua (-8 566 tavua / -93,8 %). Latausaika on pienentynyt arvosta 126,0 ms arvoon 20,8 ms (-105,2 ms / -83,5 %).

Taulukko 15. *Tapahtumien hetket ihmiskäyttäjälle lopuksi.*

Tapahtuman tyyppi	Keskiarvo	Keskihajonta
DOMContentLoaded-tapahtuma	294,1 ms	11,5 ms
Load-tapahtuma	303,3 ms	9,9 ms
Lataaminen päättyy	328,4 ms	9,3 ms

Pelkän XHTML-dokumentin lataamiseen kuluva aika on nyt 108,7 ms, kokonaisuuden ollessa 328,4 ms eli sen osuus on nyt 33,1 % entisen 51,5 %:n sijaan (-18,4 %-yks). Ladattavien tiedostojen kokonaismäärä on pienentynyt arvosta 947 905 tavua arvoon 190 333 tavua (-757 572 tavua / -79,9 %). Latausaikojen summa on pienentynyt arvosta 1 683,1 ms arvoon 578,4 ms (-1 104,7 ms / -65,6 %).

Resursseja pystytään nyt lataamaan samanaikaisesti mahdollisimman tehokkaasti, koska niiden lukumäärä ei ylitä kuutta kappaletta verkko-osoitetta kohden, joten mikään resurssi ei joudu odottamaan muiden valmistumista. Sivun lataamiseen kuluva aika selviää taulukosta 15. DOMContentLoaded-tapahtuman hetki on nopeutunut arvosta 745,6 ms arvoon 294,1 ms (-451,5 ms / -60,6 %). Keskihajonta on laskenut arvosta 75,4 ms arvoon 11,5 ms (-63,9 ms / -84,7 %). Load-tapahtuman hetki on nopeutunut arvosta 806,4 ms arvoon

303,3 ms (-503,1 ms / -62,4 %). Keskihajonta on laskenut arvosta 109,6 ms arvoon 9,9 ms (-99,7 ms / -91,0 %). Lataamisen päättymisen ajankohta on nopeutunut arvosta 837,6 ms arvoon 328,4 ms (-509,2 ms / -60,8 %). Keskihajonta on laskenut arvosta 110,6 ms arvoon 9,3 ms (-101,3 ms / -91,6 %).

Taulukko 16. *Koot ja ajat yhteensä robottikäyttäjälle lopuksi.*

Sisällön tyyppi	**Koko yhteensä**	**Aika yhteensä**
XHTML-sivu (1 kpl)	6 246 tavua	119,1 ms
Pyydetty CSS (2 kpl)	11 009 tavua	84,4 ms
Fontit edellisiin (2 kpl)	35 088 tavua	36,5 ms
Pyydetty JS (6 kpl)	126 676 tavua	281,2 ms
Sisältö edellisiin (2 kpl)	394 tavua	14,0 ms
Kuvat (6 kpl)	563 tavua	21,0 ms
Yhteensä (19 kpl)	**179 976 tavua**	**556,2 ms**

Sivun sisällön koko ja latausajat robottikäyttäjälle on koottu taulukkoon 16 ja niitä verrataan taas ihmiskäyttäjään. Ensimmäisenä on itse XHTML-sivu, jonka koko on pienentynyt arvosta 9 047 tavua arvoon 6 246 tavua (-2 801 tavua / -31,0 %). Latausaika on kasvanut arvosta 108,7 ms arvoon 119,1 ms (+10,4 ms / +9,6 %). Tästäkin voidaan taas selvästi havaita, ettei täysin luotettavia mittauksia voida tehdä normaalin Internet-yhteyden kautta. Pyydetyistä JavaScript-komentosarjoista (JS) jää pois toinen itse tehdyistä tiedostoista, joka liittyy ammattiluokitukseen. Näiden koko on pienentynyt arvosta 134 232 tavua arvoon 126 676 tavua (-7 556 tavua / -5,6 %). Latausaika on pienentynyt arvosta 320,1 ms arvoon 281,2 ms (-38,9 ms / -12,2 %). Muidenkin ryhmien latausajoissa on tietysti pieniä eroja, mutta ne johtuvat vain sattumasta, joten niiden läpikäyminen ei ole mielekästä. Ladattavien tiedostojen kokonaismäärä on pienentynyt arvosta 190 333 tavua arvoon 179 976 tavua (-10 357 tavua / -5,4 %). Latausaikojen summa on pienentynyt arvosta 578,4 ms arvoon 556,2 ms (-22,2 ms / -3,8 %).

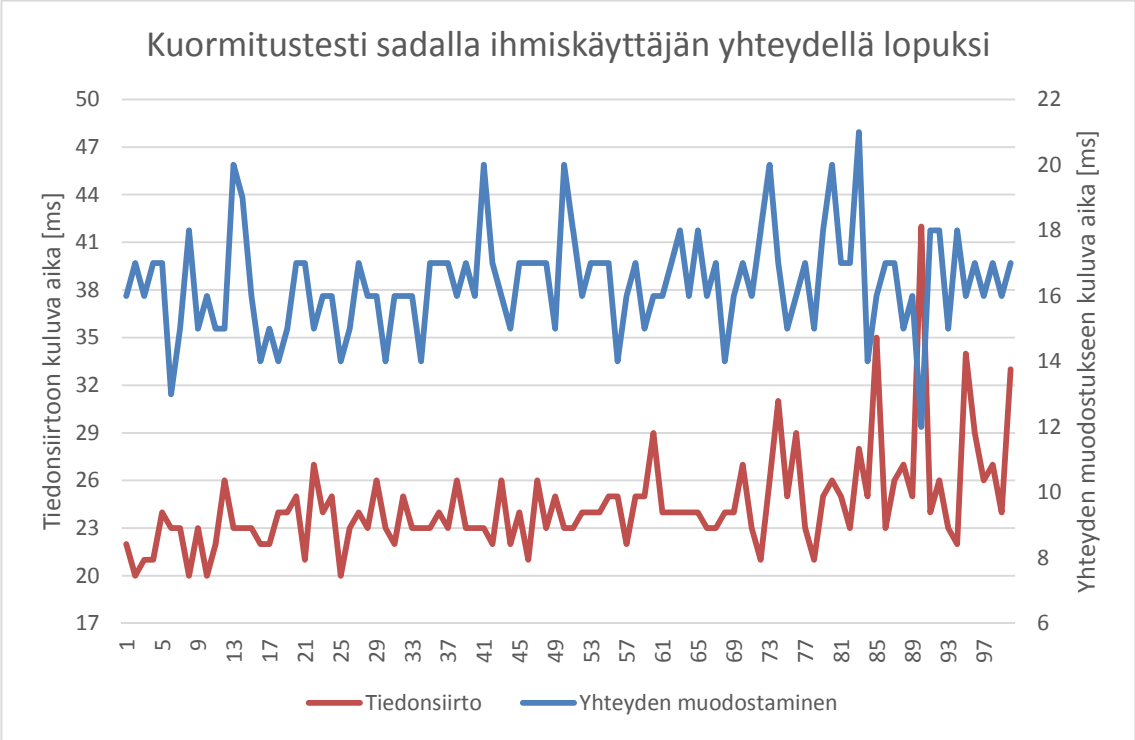
7.2 Yhteyden muodostamis- ja tiedonsiirtoajat

Tässä mittauksessa asetetaan taas ApacheBench-työkalu lähettämään jatkuvasti uusia sivupyyntöjä palvelimelle simuloiden pelkästään yhtä samanaikaista käyttäjää ja mittauksia tehdään sata kappaletta. Suurempi samanaikaisten käyttäjien määrä vääristäisi tuloksia, koska nyt halutaan selvittää vaikutus nimenomaan yhteen käyttäjään. ApacheBench lataa vain itse HTML-dokumentin, ei siihen liittyviä resursseja.

Taulukko 17. *Yhteyden muodostamisen ja tiedonsiirron kesto ihmiskäyttäjillä.*

	Minimi	**Mediaani**	**Keskiarvo**	**Maksimi**	**Keskihajonta**
Yhteyden muodostaminen	12,00 ms	16,00 ms	16,39 ms	21,00 ms	1,54 ms
Tiedonsiirto	20,00 ms	24,00 ms	24,41 ms	42,00 ms	3,19 ms

Tarkastellaan ensin ApacheBenchin antamaa kuormitustestin tulosta ihmiskäyttäjillä, jonka tulokset sadan sivupyynnön jälkeen ovat taulukon 17 mukaiset. Todetaan, että yhteyden muodostamiseen kuluva aika on edelleen pieni ja alkutilannetta vastaava. Tiedonsiirtoon kuluvan ajan mediaani on pienentynyt arvosta 169,5 ms arvoon 24,00 ms (-145,5 ms / -85,8 %) ja keskiarvo on puolestaan pienentynyt arvosta 176,5 ms arvoon 24,41 ms (-152,09 ms / -86,2 %). Myös keskihajonta on pienentynyt arvosta 18,5 ms arvoon 3,19 ms (-15,31 ms / -82,8 %). Oletettiin, että itse XHTML-sivun pienentäminen näkyisi hyvin tässä testissä ja todetaan niin myös tapahtuneen.



Kuva 14. *ApacheBenchin antamat tulokset kuvaajana ihmiskäyttäjällä lopuksi.*

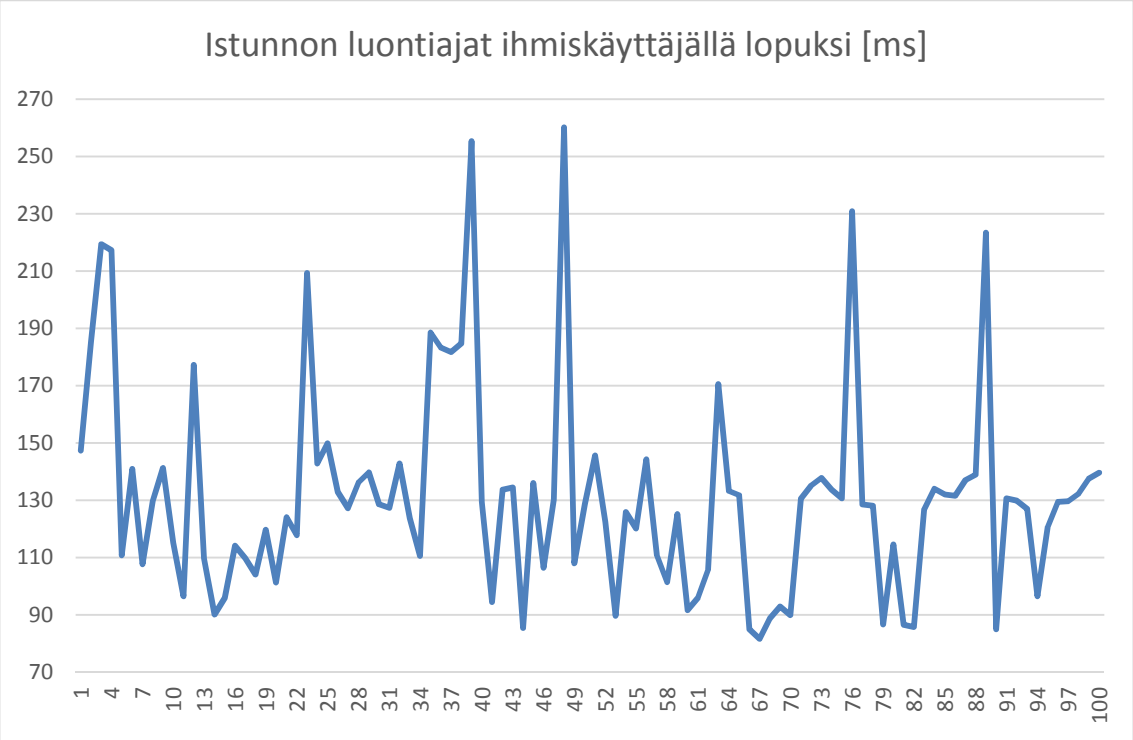
Tarkastellaan vielä kuormitustestin tuloksia kuvaajasta, jotta voidaan arvioida sitä, kuinka luotettavia taulukkoon 17 lasketut tunnusluvut ovat. Kuten kuvasta 14 voidaan todeta, että yhteyden muodostamisnopeudessa on samanlaista vaihtelua noin kymmenen millisekunnin välillä kuten aiemminkin. Tiedonsiirron nopeus näyttää aluksi pysyvän hyvin vakiona, mutta loppuun tulee muutama suurempi piikki. Arvoja katsomalla huomataan kuitenkin, että huippuarvo on kuitenkin alle 18 millisekuntia keskiarvon yläpuolella. Nopeutuminen on ollut niin huomattavaa, että pienikin muutos Internet-yhteyksissä, näkyy näissä ajoissa jo suurena hyppynä kuvaajassa.

Taulukko 18. *Yhteyden muodostamisen ja tiedonsiirron kesto roboteilla.*

	Minimi	Mediaani	Keskiarvo	Maksimi	Keskihajonta
Yhteyden muodostaminen	13,00 ms	17,00 ms	16,51 ms	22,00 ms	1,82 ms
Tiedonsiirto	12,00 ms	13,00 ms	13,83 ms	28,00 ms	2,26 ms

ApacheBenchin antamat kuormitustestin tulokset robottikäyttäjillä sadan sivupyynnön jälkeen on koottu taulukkoon 18. Yhteyden muodostamisajat eivät ole juurikaan muuttuneet, mutta sen sijaan robottikäyttäjälle sivustolla ei esimerkiksi näytetä ollenkaan ammatti- ja aluevalitsimia, joten etusivun latauskoko on myös pienempi. Edellisen testin dokumentin koko oli 28 667 tavua ja tässä se on 15 041 tavua eli 13 626 tavua ja 47,5 % pienempi. Tiedonsiirtoon kuluvan ajan mediaani on pienentynyt ihmiskäyttäjän arvosta 24 ms arvoon 13 ms (-11 ms / -45,8 %) ja keskiarvo on puolestaan pienentynyt arvosta 24,41 ms arvoon 13,83 ms (-10,58 ms / -43,3 %). Myös keskihajonta on pienentynyt arvosta 3,19 ms arvoon 2,26 ms (-0,93 ms / -29,2 %).

7.3 Istunnon luontiajat



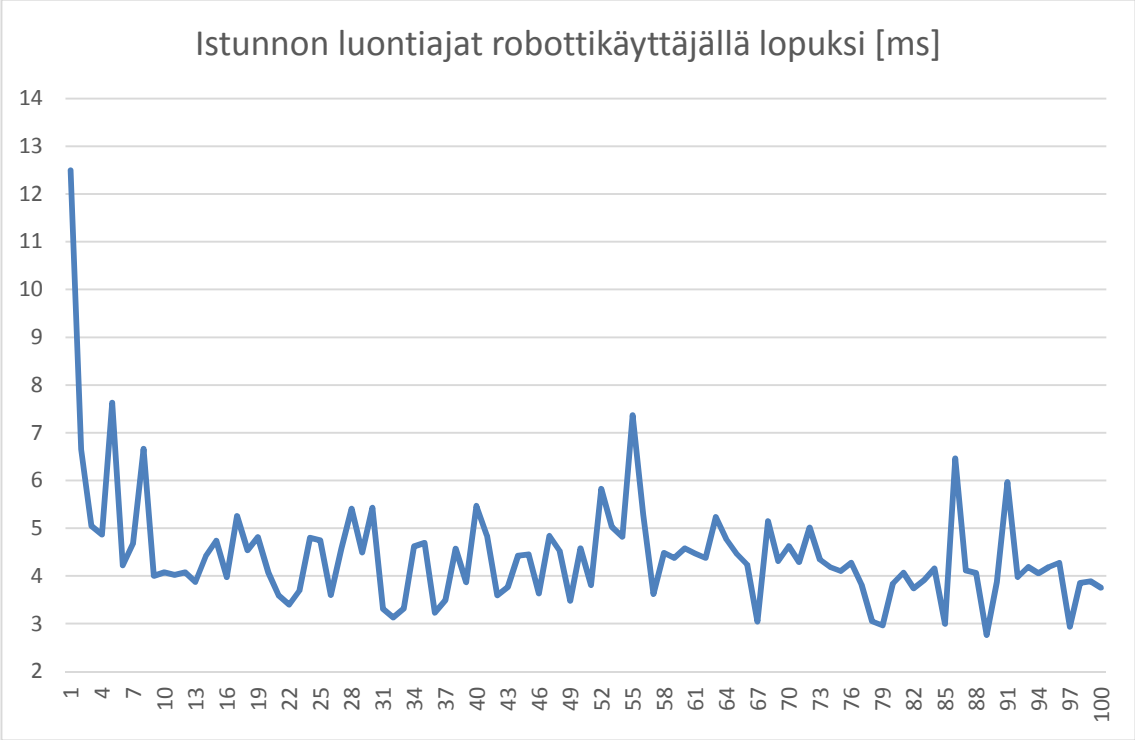
Kuva 15. *Istunnon luontiajat ihmiskäyttäjällä lopuksi.*

Taulukko 19. *Istunnon luontiaika ihmiskäyttäjällä lopuksi.*

	Minimi	Mediaani	Keskiarvo	Maksimi	Keskihajonta
Istunnon luontiaika	82 ms	129 ms	131 ms	260 ms	36 ms

Istunnon luomiseen kuluvat ajat ihmiskäyttäjällä muutosten jälkeen selviävät kuvasta 15. Tämä tulos on tallennettu taas samasta ApacheBench-kuormitustestistä, joka tehtiin alaluvussa 7.2. Tuloksista lasketaan tunnusluvut taulukkoon 19 ja lopputulokseksi saadaan vaihteluväli 82 - 260 ms sekä keskiarvoksi 131 ms ja keskihajonnaksi 36 ms. Istunnon luontiaikojen mediaani pienentyi arvosta 2019 ms arvoon 129 ms (-1890 ms / -93,6 %). Keskiarvo vastaavasti pienentyi arvosta 2022 ms arvoon 131 ms (-1891 ms / -93,5 %).

Istunnon luontiajoissa keskiarvo vastaa hyvin mediaania, josta kertoo myös pieni keskihajonta suhteessa keskiarvoon. Keskihajonta pienentyi arvosta 95 ms arvoon 36 ms (-59 ms / -62,1 %). Tavoitteeksi asetettu 200 ms saavutettiin.



Kuva 16. *Istunnon luontiajat robottikäyttäjällä lopuksi.*

Taulukko 20. *Istunnon luontiaika robottikäyttäjällä lopuksi.*

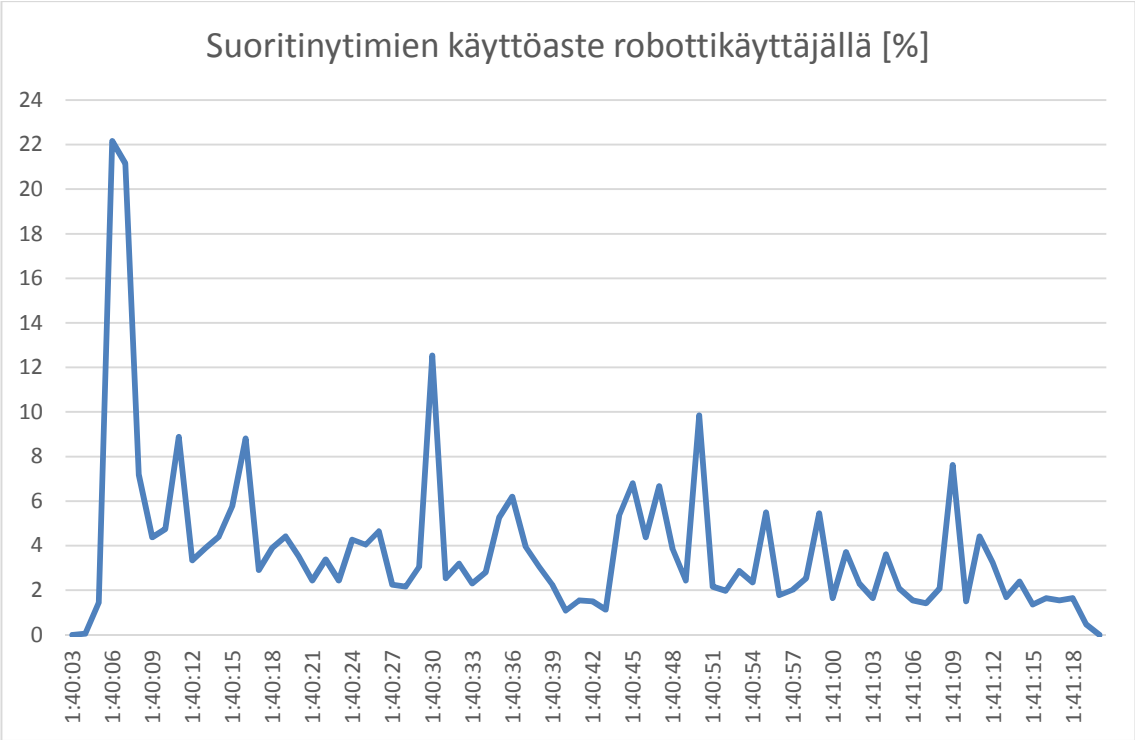
	Minimi	Mediaani	Keskiarvo	Maksimi	Keskihajonta
Istunnon luontiaika	2,76 ms	4,28 ms	4,45 ms	12,50 ms	1,19 ms

Istunnon luomiseen kuluvaa aikaa saatiin tiputettua vielä lisää edelliseen verrattuna, jos istuntoa luo robottikäyttäjä. Robottikäyttäjälle sivustolla ei esimerkiksi näytetä ollenkaan ammatti- ja aluevalitsimia, joten myöskään niihin liittyvää alustuskoodia ei tarvitse suorittaa. Istunnon luontiajat robottikäyttäjällä muutosten jälkeen selviävät kuvasta 16. Tämä tulos on tallennettu taas samasta ApacheBench-kuormitustestistä, joka tehtiin alaluvussa 7.2. Tuloksista lasketaan tunnusluvut taulukkoon 20 ja lopputulokseksi saadaan vaihteluväli 2,76 - 12,50 ms sekä keskiarvoksi 4,45 ms ja keskihajonnaksi 1,19 ms. Istunnon luontiaikojen mediaani ihmiskäyttäjään verrattuna pienentyi arvosta 129 ms arvoon 4,28 ms (-124,72 ms / -96,7 %). Keskiarvo vastaavasti pienentyi arvosta 131 ms arvoon 4,45 ms (-126,55 ms / -96,6 %). Myös keskihajonta pienentyi arvosta 36 ms arvoon 1,19 ms (-34,81 ms / -96,7 %). Olen tyytyväinen, että robottikäyttäjän istunnon luontiaika pienentyi vielä entisestään ihmiskäyttäjän istunnon luontiaikaan verrattuna ja myös tässä päästiin toivottuun kymmenesosaluokkaan.

7.4 Suoritinytimien kuormittuminen

Kuva 17. *Suoritinytimien käyttöaste ihmiskäyttäjällä lopuksi.*

Tarkastellaan suoritinytimien yhteenlaskettua käyttöastetta ihmiskäyttäjillä muutosten jälkeen kuvasta 17. Näytteitä on otettu jälleen sekunnin välein testin kestäessä 57 sekuntia. Sivupyyntöjä lähetetään testin aikana 250 kappaletta. Kuormitustestin aloitus saa aikaan heti alussa suorittimien käyttöasteeseen yhden piikin ja kaksi muuta liittyvät Javan laajempaan roskienkeruuseen. Huippuarvoksi muodostuu 25,28 %. Visuaalisesti havainnoiden voidaan todeta, että keskiarvo vastannee melko hyvin normaalitilannetta. Testin lopuksi käyttöaste putoaa luonnollisesti nollaan. Koko testijakson mediaaniarvo on 6,79 % ja keskiarvo puolestaan 8,16 %. Keskihajonnaksi muodostuu 5,18 %. Jokainen roskienkeruun sykli saa aikaan käyttöasteen kasvamisen, siksi kuvaajasta muodostuu sahalaitainen. Kun lukuja verrataan alkutilanteeseen, niin huippuarvo on kasvanut 8,39 %-yks, mediaani on kasvanut 0,54 %-yks, keskiarvo on kasvanut 1,56 % -yks ja keskihajonta on kasvanut 3,36 %-yks. Tulosten perusteella voidaan todeta, että huomattavasti pienemmästä sivusta huolimatta pakkauksen käyttöönotto lisää suoritinkuormaa hieman, mutta ei mielestäni liikaa. Täsmällisten ja tarkkojen tulosten antaminen onkin vaikeata.



Kuva 18. *Suoritinytimien käyttöaste robottikäyttäjällä lopuksi.*

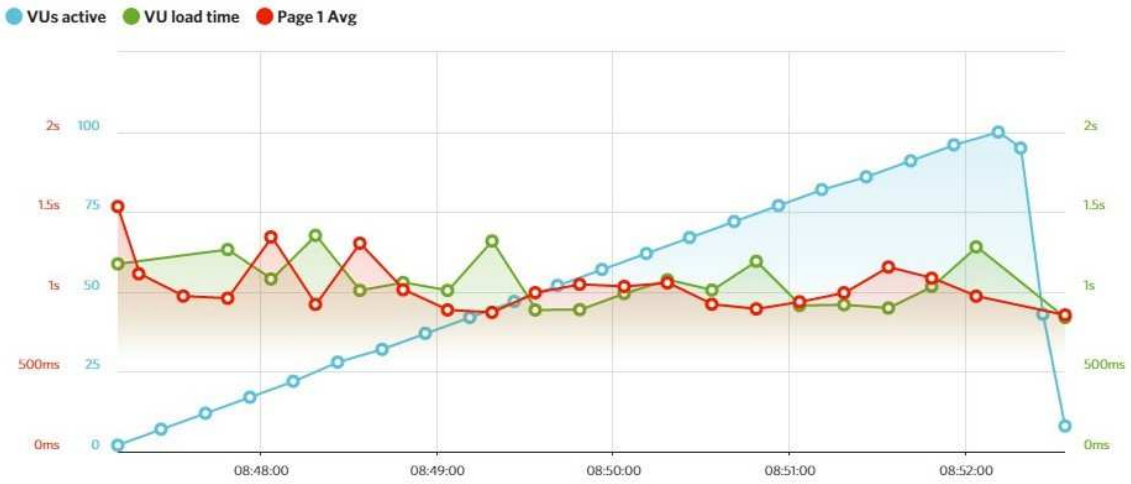
Tarkastellaan suoritinytimien yhteenlaskettua käyttöastetta robottikäyttäjillä muutosten jälkeen kuvasta 18. Näytteitä on otettu jälleen sekunnin välein testin kestäessä 77 sekuntia. Sivupyyntöjä lähetetään testin aikana tuhat kappaletta. Kuormitustestin aloitus saa aikaan heti alussa suorittimien käyttöasteeseen suurimman huippuarvon 22,16 %. Sen jälkeen käyttöaste näyttäisi hieman vakiintuvan, ja loput piikit johtuvatkin Javan roskienkeruusta. Visuaalisesti havainnoiden voidaan todeta, että keskiarvo vastannee melko hyvin normaalitilannetta. Testin lopuksi käyttöaste putoaa luonnollisesti nollaan. Koko testijakson mediaaniarvo on 2,89 % ja keskiarvo puolestaan 3,88 %. Keskihajonnaksi muodostuu 3,67 %. Kun lukuja verrataan ihmiskäyttäjien lukuihin, niin huippuarvo on pienentynyt 3,12 %-yks, mediaani on pienentynyt 3,90 %-yks, keskiarvo on pienentynyt 4,28 % -yks ja keskihajonta on pienentynyt 1,51 %-yks. Tulosten perusteella voidaan todeta, että robottikäyttäjien huomioon ottaminen auttaa myös todella hyvin pitämään suoritinkuorman kurissa pakkauksen käyttöönotosta huolimatta.

7.5 Load Impact -kuormitustesti

Suurempaa rasitusta simuloidaan edelleen ilmaisella Load Impact -kuormitustestillä, jossa simuloitujen käyttäjien määrä kasvaa viiden minuutin aikana sataan yhtäaikaiseen käyttäjään. Tässä testissä yksi käyttäjä tekee useita sivupyyntöjä ja lataa myös ulkoisia resurssitiedostoja. Yhteensä viiden minuutin aikana ihmiskäyttäjälle siirtyy 77,77 megatavua dataa, pyyntöjä tehdään 3 368 kappaletta ja etusivu latautuu 561 kertaa minimiajan ollessa 802,48 ms. Robottikäyttäjällle puolestaan siirtyy 54,51 megatavua dataa, pyyntöjä tehdään 2 756 kappaletta ja etusivu latautuu 551 kertaa minimiajan ollessa 732,13 ms.

Rasitusgeneraattorin sijainti oli edelleen molemmissa testeissä Ashburn, Virginia (VA), Yhdysvallat (USA). Luvuista voidaan jo päätellä, että tällainen pitkän matkan yhteys on herkempi Internetin muulle liikenteelle ja vaikka robottikäyttäjä lataa resursseja vähemmän, etusivu ei silti lataudu useampaa kertaa eikä paras aikakaan ole kovin paljoa nopeampi.

Kuva 19. *Yhtäaikaisten ihmiskäyttäjien kokemat latausajat lopuksi.*



Kuva 20. *Yhtäaikaisten robottikäyttäjien kokemat latausajat lopuksi.*

Sivusto piirtää rasitustesteistä myös automaattisesti kuvaajat, joista näkyy simuloitujen yhtäaikaisten käyttäjien määrä, käyttäjän kokema viive sekä sivun latausaika. Kuvasta 19 nähdään selvästi, miten viive ja latausaika pysyvät likimain vakiona aina sataan käyttäjään asti entisen 30 sijaan. Tämä rasitustesti osoittaa nyt selvästi, miten ForeAmmatti pystyy jatkossa käsittelemään vähintään kolminkertaisen määrän yhtäaikaisia käyttäjiä entiseen verrattuna. Tavoitteeksikin asetettu punaisen ja vihreän viivan pysyminen vakiona ainakin 50 käyttäjään asti täytettiin helposti.

Kuvasta 20 nähdään, miten Internetin ruuhkaisuus aiheuttaa nyt heittelyä tuloksiin, vaikka ihmiskäyttäjien testistä ei enää jäänyt parannettavaa, jonka pystyisi tällä ilmaisella

testillä havainnoimaan. Kuitenkin yksi suuri etu oli huomattavasti pienempi roskienkeruun aktiivisuus Javassa, koska robottikäyttäjille ei lasketa sellaisia tietoja, joita ne eivät kuitenkaan käyttäisi tai tarvitsisi. Tämä tuli erityisen hyvin esille tässä kuormitustestissä.

7.6 Lopputuloksen yhteenveto

ForeAmmatin etusivu koostuu nyt ihmiskäyttäjän 20 ja robottikäyttäjän 19 resurssista entisen 29 resurssin sijaan. Oman palvelimen ulkopuolelta ladataan nyt yhdeksän resurssia entisen 14 resurssin sijaan. Kaikki muut oman palvelimen resurssit käyttävät nykyään GZIP-pakkausta paitsi 509 tavun kokoinen favicon-ikoni. Tarkastellaan vielä lopuksi itse XHTML-sivun sisältöä, joka on luultavasti pienentynyt vielä hieman lisää ammatti- ja aluevalitsimien optimoinnin jälkeen. Koko sivun merkkimäärä on 23 971 merkkiä.

- Ammatti- ja aluevalitsimet yhteensä, 14 070 merkkiä ≈ 58,7 %
- josta HTML-koodia sekä tekstiä, 7 892 merkkiä ≈ 56,1 %
- ja JavaScript-koodia, 6 178 merkkiä ≈ 43,9 %
- josta ammattinimikehakua varten. 1 313 merkkiä ≈ 21,3 %

Koko sivun merkkimäärä oli aluksi 411 140 merkkiä, valitsimien optimoinnin jälkeen 55 222 merkkiä ja lopuksi 23 971 merkkiä (-387 169 merkkiä / -94,2 %). Valitsimien merkkimäärä oli aluksi 373 212 merkkiä, valitsimien optimoinnin jälkeen 17 294 merkkiä ja lopuksi 14 070 merkkiä (-359 142 merkkiä / -96,2 %). Valitsimien sisältämää HTML-koodia sekä tekstiä oli aluksi 116 885 merkkiä, valitsimien optimoinnin jälkeen 8 318 merkkiä ja lopuksi 7 892 merkkiä (-108 993 merkkiä / -93,2 %). Valitsimien sisältämää JavaScript-koodia oli aluksi 256 327 merkkiä, valitsimien optimoinnin jälkeen 8 976 merkkiä ja lopuksi 6 178 merkkiä (-250 149 merkkiä / -97,6 %). Ammattinimikehakua varten olevaa JavaScript-koodia oli aluksi 248 735 merkkiä, valitsimien optimoinnin jälkeen 1 642 merkkiä ja lopuksi 1 313 merkkiä (-247 422 merkkiä / -99,5 %).

Robottikäyttäjälle jaettavan XHTML-sivun merkkimäärä on 14 943 merkkiä (-9 028 merkkiä / -37,7 %). Ammatti- ja aluevalitsimet korvaavan osuuden merkkimäärä on 540 merkkiä (-13 530 merkkiä / -96,2 %). Tässä osuudessa HTML-koodia sekä tekstiä on 394 merkkiä (≈ 73,0 %) sekä JavaScript-koodia 146 merkkiä (≈ 27,0 %).

8. YHTEENVETO

Lopuksi kerrataan työn tärkeimmät saavutukset ja ne muutokset, jotka vaikuttavat eniten loppukäyttäjälle näkyvään käyttökokemukseen. Näissä tuloksissa ei enää esitetä robottikäyttäjään vaikuttavia tuloksia, koska ne löytyvät luvusta 7.

Taulukko 21. *Etusivun latauskokojen, -aikojen sekä tapahtumien yhteenveto.*

	Alkutilanne	**Lopputulos**
Etusivun XHTML-tiedoston latauskoko	420 998 tavua	9 047 tavua
Etusivun XHTML-tiedoston latausaika	431,7 ms	108,7 ms
Etusivun latauskoko yhteensä	947 905 tavua	190 333 tavua
Etusivun näkyviin ilmestymisen hetki (DOMContentLoaded-tapahtuma)	745,6 ms ± 75,4 ms	294,1 ms ± 11,5 ms
Etusivun valmiilta näyttämisen hetki (Load-tapahtuma)	806,4 ms ± 109,6 ms	303,3 ms ± 9,9 ms

Tärkeimmät etusivun latauskokoihin ja -aikoihin sekä tapahtumiin liittyvät muutokset on koottu taulukkoon 21. Ensimmäiseltä riviltä löytyy itse etusivun XHTML-tiedoston latauskoko, toiselta sen latausaika ja kolmannelta koko etusivun yhteislatauskoko mukaan lukien kaikki ulkoiset resurssit. Neljänneltä riviltä löytyvä DOMContentLoaded-tapahtuman hetki kertoo sen, milloin etusivusta ilmestyy jotakin käyttäjälle näkyviin. Viidenneltä riviltä löytyvä Load-tapahtuman hetki puolestaan kertoo sen, milloin etusivu näyttää käyttäjälle valmiilta.

Taulukko 22. *ApacheBench-testin sekä istunnon luontiajan yhteenveto.*

	Alkutilanne	**Lopputulos**
Etusivun tiedonsiirtoaika ApacheBench-testissä	176,5 ms ± 18,5 ms	24,41 ms ± 3,19 ms
Istunnon luontiaika	2 022 ms ± 95 ms	131 ms ± 36 ms

Taulukon 22 ensimmäiseltä riviltä löytyy ApacheBench-testin raportoima etusivun tiedonsiirtoaika sekä toiselta riviltä edellisen kanssa samaan aikaan ohjelmakoodista mitattu istunnon luontiaika.

Taulukko 23. *Load Impact -kuormitustestin yhteenveto.*

	Alkutilanne	**Lopputulos**
Etusivun onnistuneita latauksia viiden minuutin aikana	241 kappaletta	561 kappaletta
Etusivun nopein latautuminen viiden minuutin aikana	3,71 sek	802,48 ms
Simuloitujen yhtäaikaisten käyttäjien määrä käyttäjän kokeman viiveen ja latausajan pysyessä muuttumattomana	noin 30 käyttäjää	väh. sata käyttäjää

Load Impact -kuormitustestin tulosten yhteenveto on koottu taulukkoon 23. Ensimmäiseltä riviltä löytyy etusivun onnistuneiden latausten määrä viiden minuutin testijakson aikana. Toinen rivi kertoo kyseisen sivuston raportoiman minimiajan, jolla etusivu on testin aikana onnistuneesti latautunut. Kolmas rivi puolestaan kertoo, montako testin simuloimaa yhtäaikaista käyttäjää ForeAmmatti-palvelu kestää tämän äärimmäisen rasitustestin aikana ilman, että käyttäjän kokema viive ja latausaika lähtevät kasvamaan.

Taulukko 24. *Etusivun ja valitsimien merkkimäärien yhteenveto.*

	Alkutilanne	**Lopputulos**
Etusivun yhteismerkkimäärä	411 140 merkkiä	23 971 merkkiä
Ammatti- ja aluevalitsimien merkkimäärä	373 212 merkkiä	14 070 merkkiä
Valitsimien HTML-koodia sekä tekstiä	116 885 merkkiä	7 892 merkkiä
Valitsimien JavaScript-koodia	256 327 merkkiä	6 178 merkkiä
Ammattinimikehaun JavaScript-koodia	248 735 merkkiä	1 313 merkkiä

Etusivun ja sen sisältämien ammatti- ja aluevalitsimien merkkimäärien yhteenveto on koottu taulukkoon 24. Ensimmäinen rivi kertoo etusivun yhteismerkkimäärän ja toinen valitsimien yhteismerkkimäärän. Kolmas rivi kertoo valitsimien sisältämän HTML-koodin sekä tekstin merkkimäärän, neljäs valitsimien sisältämän JavaScript-koodin merkkimäärän ja viides ammattinimikehaun merkkimäärän osuuden JavaScript-koodin merkkimäärästä.

Lopuksi listaan vielä lyhyesti tehtyjen optimointien haittapuolia. Jos ja kun ammattiluokitukseen tulee muutoksia, pitää ne muuttaa myös kovakoodattuihin JavaScript-tiedostoihin. Näitä muutoksia tulee kuitenkin erittäin harvoin, korkeintaan muutaman kerran vuodessa, eikä työ itsessään ole suuri. Lisäksi tietysti pitää hallita kaikista CSS- ja JavaScript-tiedostoista useammat versiot, koska itselläni pitää olla myös minimoimattomat versiot, joihin uudet muutokset tehdään. Meillä nämä tiedostot eivät onneksi muutu kovin usein, korkeintaan parin viikon välein, joten sekään vaiva ei ole kovin suuri. Lisäysten ja muutosten jälkeen tiedostot pitää toki minimoida taas uudelleen ja päivittää tiedostonimien aikaleimat sekä näitä resursseja kutsuvat rivit.

Subjektiivisestikin arvioituna ForeAmmatti toimii muutosten jälkeen huomattavasti nopeammin, myös nopealla yhteydellä, ja toki mittauksetkin vahvistavat asian. Pidän työstä saatuja hyötyjä sekä siitä kertyneitä saavutuksia erittäin merkittävinä, joiden jälkeen tuotteemme on entistäkin parempi ja käyttäjäystävällisempi.

Lopuksi tiivistän vielä hieman yleisemmällä tasolla muutaman optimointivinkin, joiden avulla itse pääsin esitettyihin tuloksiin. Tärkeintä on tunnistaa suurimmat ongelmakohdat ja aloittaa optimointi niistä. Tärkeää on myös kokeilla useampaa kuormitustesteriä tai ruuhkauttamispalvelua, ja tunnistaa niiden antamat tulokset. Optimointi onkin jatkuvasti iteroituva prosessi, jossa saat uusia ideoita aina tasaisin väliajoin. Kun pääset tiettyyn pisteeseen asti asiakaspuolella, on aika siirtyä palvelinpuolelle ja päinvastoin.

Yritä siirtää välimuistiin kelpaamattoman XHTML- tai HTML-sivun sisällä mahdollisimman vähän tekstiä ja vain välttämättömin sisältö. Voit siirtää esimerkiksi erilaisten listojen muodostukseen tarvittavat tunnisteet tai ID-arvot yhdessä JavaScript-vektorissa ja antaa käyttäjän selaimen muodostaa lopputulos. Mikäli tekstit ovat aina samoja, mutta vain erilaisessa järjestyksessä, voit tallentaa ne erilliseen JavaScript-tiedostoon, joka voidaan tallentaa välimuistiin. Siirrä sivun lataushetkellä vain se data, jota kaikki käyttäjät tarvitsevat. Muu data, hakutulokset, lisäsisällöt ynnä muut haet asynkronisesti palvelimelta vain, jos käyttäjä niitä pyytää.

XHTML-tiedoston tarkan syntaksin takia toimivaa minimointikoodia voi olla hankala löytää, mutta ainakin sisennysten poistaminen säännöllisen lausekkeen avulla on helppoa ja sekin säästää jo merkkejä. CSS- ja JavaScript-tiedostot ovat helpompia, koska niiden minimointiin löytyy useitakin erilaisia sivustoja ja työkaluja, joita voit kokeilla parhaan lopputuloksen löytämiseksi. Etsi ulkopuolisille resurssitiedostoille sopiva lataushetki tarpeesi mukaan riippuen siitä, onko hyödyllisempää ladata ne heti vai vasta myöhemmin. Valmiiden kirjastojen kanssa käytä aina minimoituja resursseja ja valitse niiden selaintuki käyttäjiesi mukaan. Mikäli mahdollista, kustomoi valmiit kirjastot omien tarpeidesi mukaan, äläkä pidä mukana tarpeettomia tai vanhentuneita ominaisuuksia.

Hyödynnä valmiit muiden tekemät ominaisuudet, kuten pakkaukset, välimuistit ja välityspalvelimet tarpeesi mukaan. Pysy kehityksen mukana, ja seuraa tulevia uusia tekniikoita, kuten vaikkapa HTTP/2-protokollaa. Varaudu siihen, että tällä hetkellä tekemäsi optimoinnit eivät välttämättä ole enää tehokkaita puolen vuoden päästä. Muista kuitenkin, että jokainen pienikin askel on aina askel kohti nopeampaa ja luotettavampaa sivustoa sekä tyytyväisempää käyttäjää.

LÄHTEET

[1] Ammattiluokitus, Työministeriö, 2005, s. 237-238. Saatavissa (viitattu 30.10.2015): http://www.te-palvelut.fi/te/fi/pdf/ammattiluokitus.pdf

[2] Ammattiluokitus 2010, Tilastokeskus, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.stat.fi/meta/luokitukset/ammatti/001-2010/

[3] Asynchronous Loading – Google Maps JavaScript API, Google Developers, verkkosivu. Saatavissa (viitattu 27.7.2015): https://developers.google.com/maps/documentation/javascript/examples/map-simple-async

[4] Kalid Azad, How To Optimize Your Site With GZIP Compression, verkkosivu. Saatavissa (viitattu 30.10.2015): http://betterexplained.com/articles/how-to-optimize-your-site-with-gzip-compression

[5] Kalid Azad, How To Optimize Your Site With HTTP Caching, verkkosivu. Saatavissa (viitattu 30.10.2015): http://betterexplained.com/articles/how-to-optimize-your-site-with-http-caching

[6] Base64 Image Encoder, Dominik Hanke, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.base64-image.de/

[7] Compare It!, Grig Software, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.grigsoft.com/wincmp3.htm

[8] CSS Compressor, verkkosivu. Saatavissa (viitattu 30.10.2015): http://csscompressor.com/

[9] CSS Usage - Add-ons for Firefox, Mozilla, verkkosivu. Saatavissa (viitattu 30.10.2015): https://addons.mozilla.org/en-US/firefox/addon/css-usage/

[10] CSS3 Browser Support Reference, W3Schools, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.w3schools.com/cssref/css3_browsersupport.asp

[11] DOMContentLoaded – Event reference, Mozilla Developer Network, verkkosivu. Saatavissa (viitattu 30.10.2015): https://developer.mozilla.org/en-US/docs/Web/Events/DOMContentLoaded

[12] Marius Ducea, ApacheBench with Mod_GZip, Mod_Deflate – MDLog:/sysadmin, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.ducea.com/2006/10/28/apachebench-with-mod_gzip-mod_deflate/

[13] T. Elonen, Suorituskykyinen verkkosivu sisällönhallintajärjestelmän avulla, Metropolia ammattikorkeakoulu, insinöörityö, 2011, 60 s. Saatavissa: http://www.theseus.fi/bitstream/handle/10024/32688/tero_elonen.pdf

[14] Execute JavaScript before and after the f:ajax listener is invoked, Stack Overflow, verkkosivu. Saatavissa (viitattu 30.10.2015): http://stackoverflow.com/a/24926781

[15] ExpiresFilter - Apache Tomcat 8.0 API Documentation, Apache Software Foundation, verkkosivu. Saatavissa (viitattu 30.10.2015): https://tomcat.apache.org/tomcat-8.0-doc/api/org/apache/catalina/filters/ExpiresFilter.html

[16] Explicit loading of jsf.js - Mastering JavaServer Faces 2.2, Safari, verkkosivu. Saatavissa (viitattu 30.10.2015): https://www.safaribooksonline.com/library/view/mastering-javaserver-faces/9781782176466/ch07s11.html

[17] Firebug – Web Development Evolved, Mozilla, verkkosivu. Saatavissa (viitattu 30.10.2015): http://getfirebug.com/

[18] ForeAmmatti – Usein kysyttyä, Foredata Oy, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.foreammatti.fi/ukk

[19] Google Chrome release history, Wikipedia, verkkosivu. Saatavissa (viitattu 30.10.2015): https://en.wikipedia.org/wiki/Google_Chrome_release_history

[20] Highcharts – Download, Highsoft AS, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.highcharts.com/download

[21] Highcharts Product, Highsoft AS, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.highcharts.com/products/highcharts

[22] History of Firefox, Wikipedia, verkkosivu. Saatavissa (viitattu 30.10.2015): https://en.wikipedia.org/wiki/History_of_Firefox

[23] History of Opera web browser, Wikipedia, verkkosivu. Saatavissa (viitattu 30.10.2015): https://en.wikipedia.org/wiki/History_of_the_Opera_web_browser

[24] HTTP/2, IETF HTTP Working Group, verkkosivu. Saatavissa (viitattu 22.11.2015): https://http2.github.io/

[25] HTTP/2 for Web Application Developers, NGINX, syyskuu 2015, 16 s. Saatavissa: https://www.nginx.com/wp-content/uploads/2015/09/NGINX_HTTP2_White_Paper_v4.pdf

[26] Internet Explorer 9, Wikipedia, verkkosivu. Saatavissa (viitattu 30.10.2015): https://en.wikipedia.org/wiki/Internet_Explorer_9

[27] Internet Explorer versions, Wikipedia, verkkosivu. Saatavissa (viitattu 30.10.2015): https://en.wikipedia.org/wiki/Internet_Explorer_versions

[28] JavaScript Compressor, verkkosivu. Saatavissa (viitattu 30.10.2015): http://javascript-compressor.com/

[29] JavaScript Cruncher, Webmasters Cavern, verkkosivu. Saatavissa (viitattu 5.8.2015): http://www.webmasters-cavern.com/tools/jscruncher.htm

[30] Jukka K. Korpela, HTML5 – Uudet ominaisuudet, WSOYpro Oy, Docendo-tuotteet, 2011, s. 42.

[31] Load Impact – On Demand Load Testing for Developers & Testers, verkkosivu. Saatavissa (viitattu 30.10.2015): https://loadimpact.com/

[32] PageSpeed Tools, Google Developers, verkkosivu. Saatavissa (viitattu 30.10.2015): https://developers.google.com/speed/pagespeed/

[33] Ravintola- ja suurtaloustyöntekijät – Ammattiluokitus 2010, Tilastokeskus, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.stat.fi/meta/luokitukset/ammatti/001-2010/5120.html

[34] Safari version history, Wikipedia, verkkosivu. Saatavissa (viitattu 30.10.2015): https://en.wikipedia.org/wiki/Safari_version_history

[35] A Simple Performance Comparison of HTTPS, SPDY and HTTP/2, HttpWatch, verkkosivu. Saatavissa (viitattu 30.10.2015): http://blog.httpwatch.com/2015/01/16/a-simple-performance-comparison-of-https-spdy-and-http2/

[36] SPDY Protocol, Network Working Group – Google Inc, verkkosivu. Saatavissa (viitattu 22.11.2015): http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

[37] Talousjohtajat – Ammattiluokitus 2010, Tilastokeskus, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.stat.fi/meta/luokitukset/ammatti/001-2010/1211.html

[38] Text-size-adjust – CSS, Mozilla Developer Network, verkkosivu. Saatavissa (viitattu 30.10.2015): https://developer.mozilla.org/en-US/docs/Web/CSS/text-size-adjust

[39] TinyPNG, Voormedia, verkkosivu. Saatavissa (viitattu 30.10.2015): https://tinypng.com/

[40] Toimiala Online -tietopalvelu, Työ- ja elinkeinoministeriö, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www2.toimialaonline.fi/

[41] T. Tuominen, ForeAmmatti-tietojärjestelmä, Satakunnan ammattikorkeakoulu, opinnäytetyö, 2011, 42 s. Saatavissa: https://www.theseus.fi/bitstream/handle/10024/28924/Tuominen_Tommi.pdf

[42] I. Vataja, ForeAmmatti-tietojärjestelmän tietokannat, Satakunnan ammattikorkeakoulu, opinnäytetyö, 2011, 28 s. Saatavissa: https://www.theseus.fi/bitstream/handle/10024/36201/Ilkka_Vataja_2011.pdf

[43] H. Virta, Sisällönhallintajärjestelmää käyttävän verkkosivun suorituskyvyn optimointi, Metropolia ammattikorkeakoulu, insinöörityö, 2012, 43 s. Saatavissa: http://www.theseus.fi/bitstream/handle/10024/72200/HeidiVirta_ins_final.pdf

[44] When is a CDATA section necessary within a script tag?, Stack Overflow, verkkosivu. Saatavissa (viitattu 5.8.2015): http://stackoverflow.com/a/66865

[45] WOFF – Web Open Font Format, verkkosivu. Saatavissa (viitattu 30.10.2015): http://caniuse.com/woff

[46] WOFF 2.0 – Web Open Font Format, verkkosivu. Saatavissa (viitattu 30.10.2015): http://caniuse.com/woff2

[47] XML DOM – CDATASection object, W3Schools, verkkosivu. Saatavissa (viitattu 30.10.2015): http://www.w3schools.com/xml/dom_cdatasection.asp